Network Coding for Speedup in Switches

by

MinJi Kim

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2007

Author .....
Department of Electrical Engineering and Computer Science
August 21, 2007

Certified by .....
Muriel Médard
Esther and Harold E. Edgerton Associate Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Network Coding for Speedup in Switches

by

MinJi Kim

Submitted to the Department of Electrical Engineering and Computer Science
on August 21, 2007, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Network coding, which allows mixing of data at intermediate network nodes, is known to increase the throughput of networks. In particular, it is known that linear network coding in a crossbar switch can sustain traffic patterns that cannot be served if network coding were not allowed. Thus, network coding leads to a larger rate region in a multicast crossbar switch.

This thesis quantifies the gain in rate region in a multicast crossbar switch in terms of speedup. We present a graph theoretic upper bound on speedup needed to achieve 100% throughput in a multicast switch using network coding. By bounding speedup, we show the equivalence between network coding and speedup in multicast switches - *i.e.* network coding, which is usually implemented using software, can in many cases substitute speedup, which is often achieved by adding extra switch fabrics. This bound is based on an approach to network coding problems called the "enhanced conflict graph". We show that the "imperfection ratio" of the enhanced conflict graph gives an upper bound on speedup. In particular, we apply this result to $K \times N$ switches with traffic patterns consisting of unicasts and broadcasts only to obtain an upper bound of $\min(\frac{2K-1}{K}, \frac{2N}{N+1})$.

Thesis Supervisor: Muriel Médard
Title: Esther and Harold E. Edgerton Associate Professor

Acknowledgments

I would like to thank my advisor, Professor Muriel Médard, for her support and guidance which were vital to my development. Her insight continues to amaze me, and her enthusiasm motivates me to work harder. Her advice and encouragement have helped me through many difficult times, and I would like to thank her for her constant patience and understanding. She has inspired me both professionally and personally, and I look forward to continuing my study under her.

I thank everybody who has guided and helped me in the course of this work. In particular, I would like to thank Jay Kumar Sundararajan for all his helpful discussions. In many ways, he has been my mentor as well as my tutor, and I appreciate the time and effort he has invested in me.

I would also like to thank my friends, who have supported me throughout my career at MIT. First of all, I thank Marjorie Cheng, Min Deng, and Shaun J. Foley for being wonderful friends – never refusing to listen to me and always giving me strength. I also thank Shiling Seow, Jon W. Wetzel, and Jonathan M. Rhodes for being part of my past five years at MIT.

I thank my mother Kyung-Sook Jeong, my father Il-Doo Kim, and my sister Hyo-Jeong Kim for their never ending love and support, without which I would not be where I am today. I am infinitely grateful for everything they have done for me.

Finally, I would like to acknowledge all my friends and family that I did not mention here. Everybody I have met has given me invaluable lessons, and although not explicitly mentioned here, I am indebted to them all.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Network information flow problem is a field of information theory and coding theory which aims to attain the maximum information flow in a network. The network information flow is closely related to the multi-commodity flow problems and has been studied extensively owing to its wide application such as communication network and packet routing.

An information network is represented by a directed graph $\mathcal{N} = (V, E)$ where $(i, j) \in E$ if information can be sent noiselessly from node $i \in V$ to node $j \in V$. There are two special subsets S and T of V. The set S is the set of sources, which generates mutually independent streams of information or *messages*. The set T is the collection of sinks. Each sink node $t \in T$ requires some subset of the information streams from the source nodes. This is called the *multicast requirement*.

The main question in network information flow is that given a network $\mathcal{N} = (V, E)$ and a multicast requirement, is it possible to satisfy all the sink nodes? Before the notion of network coding was introduced, researchers and scientists have focused on answering this question in a *transportation network*. A transportation network is a network where each packet that enters a node can only be routed or *relayed* onto some outgoing link(s). In other words, the intermediate nodes in the network cannot modify the packets that they receive – they can only forward the packets. However, Ahlswede, Cai, Li, and Yeung [2] point out that from an information theoretic point of view, there is no reason to restrict the nodes to be just routers. In their seminal

paper [2], Ahlswede et. al. introduce the notion of *network coding*. The main idea behind network coding is to allow mixing of data at intermediate network nodes. A packet that enters a node can be duplicated or encoded before it is routed onto some outgoing link. In essence, the difference between routing and network coding is in the output functions that are allowed at the intermediate nodes.

It has been shown that this simple idea allows networks to send more information – *i.e.* achieve larger rate region. The next natural questions to ask are: What kind of networks and multicast requirements are solvable using network coding? When and how much benefit does network coding give us compared to routing? What kind of network coding - linear, vector, non-linear - should we use given a network and a multicast requirement?

Recently, there has been a lot of work to understand the advantages of network coding, and how to generate network codes that satisfy the multicast requirement of all the terminals in the network. For example, Ahlswede et al. [2] showed that with network coding, as symbol size approaches infinity, a source can mutlicast information at a rate approaching the minimum cut between the source and any receiver. Li, Yeung and Cai [10] showed that any solvable multicast network has a scalar linear solution over a sufficiently large finite field alphabet. However, there are still many unanswered questions regarding network coding and its potential. In this thesis, we study the magnitude and the frequency of the benefit we gain from using network coding in a special network – multicast switches (see Section 2.3), which will hopefully bring insight to that of general networks.

1.1 Motivation

A crossbar switch is special subset of networks, which is one of the principal architectures used to construct switches. Switches are networks of depth one – these networks only consist of terminal nodes, and the source nodes or the *inputs* are directly connected to each sink node or the *outputs*. Crossbar switches can be thought of as a matrix of switches between the inputs and the outputs of the switch. In other words,

a crossbar switch with K inputs and N outputs has a $K \times N$ matrix of intersections or places where the inputs and outputs "cross" as shown in Figure 1-1. Crossbar switches are widely used in information processing applications such as telephony and packet switching.

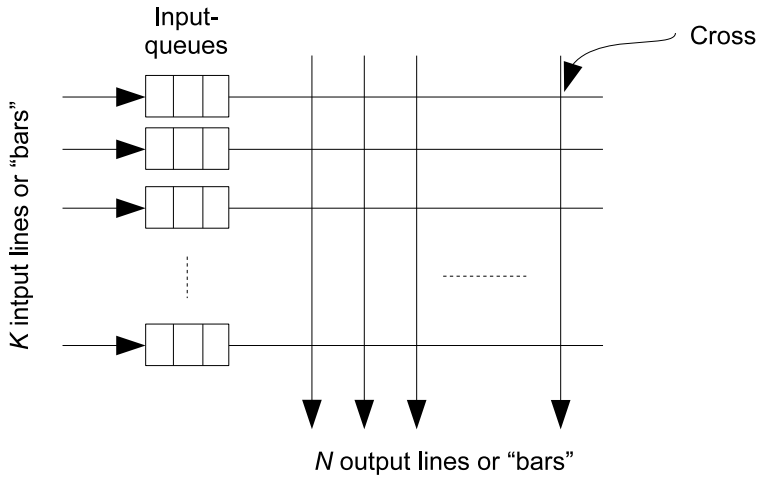

Figure 1-1: A diagram of an $K \times N$ input-queued crossbar switch

An input-queued crossbar switch is a crossbar switch with queues at the input such that incoming information or packets are queued at the inputs before it is processed. The input-queued crossbar switch has been studied extensively, especially in the context of unicast traffic, where unicast means that for a single stream of information from a source, there is one single destination sink. Therefore, a unicast traffic is a set of information streams, or a traffic pattern, which only consists of unicasts. It is known that 100% throughput can be achieved [13], in the sense that as long as no input or output is oversubscribed with more than one packet going through it per unit time, any unicast traffic can be supported without causing the queues to grow unboundedly. Therefore, if the traffic consists of unicasts only and is *admissible*, then we can serve the traffic. Here, by admissible, we mean that the total rate going through each input and output is no more than one; otherwise, there is at least one input or output with more than one packet of information traversing it, which is physically not feasible.

Unfortunately, this result does not extend to multicast flows, where a single stream of information from a source may be destined to more than one sink. The extension of the problem to multicast flows is intrinsically more difficult than that of unicast flows in the sense that 100% throughput cannot be achieved. Marsan *et al.* [12] gave a characterization of the rate region achievable in a multicast switch with *fanout splitting*. Fanout splitting is the ability to partially serve a multicast flow to only a subset of its destined outputs, and complete the service in subsequent time slots. Marsan *et al.* also defined the optimal scheduling policy. Interestingly, this work proved that unlike in the unicast case, 100% throughput cannot be achieved for multicast flows in an input-queued switch. As a result, we need to provide the multicast switch extra machinery to achieve 100% throughput.

One of the extra machineries that we can give to a multicast switch is *speedup*. Speedup allows the switch to process information faster than that of the input or output line, and is usually implemented using parallelization of hardware. For example, if the switch can internally process two packets of information in the time interval when at most one packet of information can arrive at the switch (input line rate) or leave the switch (output line rate), then the switch is said to have speedup of two. In fact, the minimum speedup needed to achieve 100% throughput is shown to grow unboundedly with the switch size for multicast traffic. It is not hard to observe that with enough speedup, a switch can achieve any traffic pattern; however, as it is in the case with most hardware features, speedup is expensive to implement and hard to change or update once the switch is installed.

Another extra machinery that we can give to a switch is linear network coding. In an input-queued crossbar switch, there are only $K + N$ nodes (the inputs and the outputs). Therefore, linear network coding is just a feature that allows the inputs to send linear combination of the packets in its queue. It is known that network coding increases throughput in a multicast switch (see Section 2.3 for more detailed discussions of the benefits of network coding in switches). The important thing to note here is that network coding, unlike speedup, can be implemented using software. Therefore, it is desirable to use network coding instead of speedup, if possible, since

it is advantageous to use software over hardware in terms of cost and extensibility.

In this thesis, we ask the question, what is the magnitude and the frequency of the benefit we gain from using network coding in multicast switches? Can we replace speedup with network coding? If not completely, then by how much?

1.2 Main Contribution and Thesis Outline

This section gives an outline of the thesis as well as the main contributions of this work. In this thesis, a switch is allowed to send linear combinations of packets waiting in the input queues, *i.e.*, they are allowed to perform linear network coding with fanout splitting. This work studies the benefit of linear network coding in a multicast switch, and quantifies the increase in throughput due to network coding in terms of speedup. By studying the benefits of network coding in terms of speedup, we show the equivalence between network coding and speedup in multicast switches - *i.e.* network coding, which is usually implemented using software, can in many cases substitute for speedup, which is often achieved by adding extra switch fabrics.

The main contributions of this paper are:

1. We show that network coding can in many cases substitute for speedup.

2. We provide a simple graph-theoretic upper bound on speedup to achieve 100% throughput.

3. We prove an upper bound on speedup of $\min\left(\frac{2K-1}{K}, \frac{2N}{N+1}\right)$ for an arbitrary $K \times N$ switch with traffic pattern restricted to unicasts and broadcasts only.

The rest of the thesis is organized as follows. Chapter 2 covers some relevant background and preliminary definitions that will be used throughout this thesis. The thesis mainly draws ideas from network coding (Section 2.1) and graph theory (Section 2.2). The thesis focuses quantifying the benefit of network coding in a special type of network, called multicast switches. Therefore, in Section 2.3, we shall give some background on multicast switches.

Chapter 3 discusses the benefits of network coding. First, we present a graph theoretical formulation of network coding in multicast swtiches. This graph theoretical formulation, called the conflict graph, translates the problem of finding the rate region of a network as well as scheduling into that of combinatorics. As a result, this formulation allows us to use tools from graph theory and combinatorics to gain insights into the benefits of network coding. It is important to note that such formulation is not available for fanout-splitting or non-network coding networks; thus, network coding not only increases throughput, but also provides an insightful formulation that brings the problem to its combinatorial essence, which determines its difficulty. These results show that network coding is advantageous.

Our next question, then, concerns the practicality of linear network coding. We know that network coding is beneficial, but when does this benefit actually manifest itself? To answer this question, in Section 3.2 and Section 3.3, we give simulation results which show that network coding improves the throughput as well as delay not only in theoretical sense but also in practical sense.

In Chapter 4, we present our main contribution. Using the graph theoretical formulation introduced in Chapter 3, we draw a connection between speedup needed to achieve 100% throughput and *imperfection ratio*, a measure of the perfectness of a graph (see Section 2.2.3). This allows us to get an upper bound on the minimum speedup required to achieve 100% throughput in a multicast switch using linear network coding. This result shows that linear network coding cannot replace speedup entirely; however, linear network coding does increase throughput and as a result replace speedup in some cases. Another important point to note is that, although our speedup results are only for networks of depth one, *i.e.* switches, our approach using conflict graphs is general.

Furthermore, we apply the relationship between imperfection ratio and speedup from Chapter 4 to switches with special traffic patterns. First, we apply it to a $2 \times N$ switch with traffic pattern consisting only of unicasts and broadcasts, and we show that speedup of at most $3/2$ is needed to achieve 100% throughput in this case. We, then, generalize this result to a $K \times N$ switch with traffic consisting only of unicasts

and broadcasts. This gives us an upper bound on speedup of $\min(\frac{2K-1}{K}, \frac{2N}{N+1})$.

Finally, in Chapter 5, we summarize the contributions of this thesis, and present a conjecture on the actual minimum speedup needed to achieve 100% throughput in a $2 \times N$ multicast switch with unicasts and broadcasts only. Furthermore, we also discuss possible line of future work related to this thesis.

Chapter 2

Background

This chapter gives an overview of the relevant work in the area of network coding (Section 2.1), graph theory (Section 2.2), and switching theory (Section 2.3).

2.1 Network Coding

Network information flow problem is a field of information theory and coding theory which aims to attain the maximum information flow in a network. The network information flow is closely related to the multi-commodity flow problems and has been studied extensively owing to its wide application such as communication network and packet routing.

A point-to-point communication network is represented by a direct graph $\mathcal{N} = (V, E)$ where $(i, j) \in E$ if information can be sent from node $i \in V$ to node $j \in V$. There are two special subsets S and T of V. The set S is the set of sources; the set T is the set of sinks. The source nodes generate mutually independent streams $X_1, X_2, ..., X_m$ of information or *messages*, where m is not constrained by $|S|$. Each information stream $X_i = (r_i, s_i, T_i)$ is said to have information rate r_i (in bits per unit time). X_i is generated at source node $s_i \in S$, and it is multicast to all sink nodes $t \in T_i \subseteq T$. This set of information streams $\mathcal{X} = \{X_1, X_2, ..., X_m\}$ is called the *multicast requirement*.

The main question in network information flow is that given $\mathcal{N} = (V, E)$ can

we satisfy the multicast requirement $\mathcal{X}$? Before the notion of network coding was introduced, researchers and scientists have focused on answering this question in a *transportation network*. A transportation network is a network where each packet that enters a node can only be routed or *relayed* onto some outgoing link(s). However, Ahlswede, Cai, Li, and Yeung [2] point out that from an information theoretic point of view, there is no reason to restrict the nodes to be just routers. In their seminal paper [2], Ahlswede et. al. introduce the notion of *network coding*. The main idea behind network coding is to allow mixing of data at intermediate network nodes. A packet that enters a node can be duplicated or encoded before it is routed onto some outgoing link. In essence, the difference between routing and network coding is in the time-varying output functions f_t that are allowed at the intermediate nodes. Note that f_t is time-varying – *i.e.* the function changes with time t. For simplicity, we shall not explicitly indicate the output function's dependency on time t and denote the output function f_t as f. However, it is important to keep in mind that f is time-varying.

In Figure 2-1, we present a node in a network. The node has three incoming messages x_1, x_2, and x_3. The output function of the node is given by f. In the case of routing, f is restricted to be $f(x_1, x_2, x_3) = x_i$ where $i \in [1, 3]$; while in the case of network coding, f can be from any arbitrary set of functions F. Therefore, we note that there can be an enormous number of different classes of network coding, depending on how we restrict the set F of output functions. For instance, routing is a particular class of network coding where $F = \{f \mid f(x_1, x_2, ..., x_n) = x_i \text{ for some } i \in [1, n]\}$.

Another special class of network coding is the *linear network coding*, which allows the output function at the intermediate nodes to be a linear combination of incoming information packets. For example, in the case of linear network coding, the output function in Figure 2-1 will be of the form $f(x_1, x_2, x_3) = c_1x_1 + c_2x_2 + c_3x_3$ for some constants c_1, c_2, c_3. We shall revisit linear network coding in Section 2.1.1

This simple idea of duplicating and encoding packets allows information flow networks to send more information (i.e. achieve larger rate region) than transportation

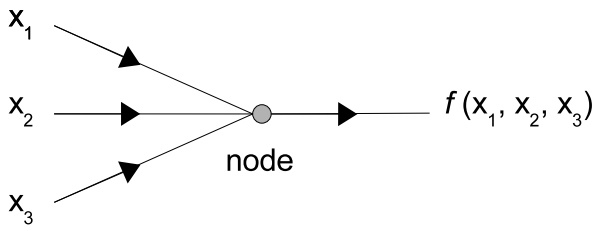

Figure 2-1: A node in a network.

networks. The intuition behind this result is that in a traditional transport network, a single packet A cannot be on a link l without preventing another packet B from using the same link l; however, with network coding, packet A can "share" the link l with packet B by assigning $A + B$ on l.

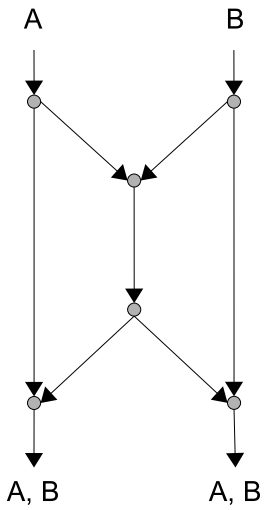

Figure 2-2: The butterfly network from [2]

The rudimentary example used to show the advantage of network coding is in Figure 2-2. The two sources are generating two information stream A and B at rate equal to 1. The two sinks want to receive both stream of information every unit time. Pure routing would fail to achieve this multicast requirement since sending one sink both A and B would entail the remaining sink only receiving just A or B but not both. However, network coding, as shown in Figure 2-3, can satisfy both sinks.

Transmitting $A + B$ on the middle link, where '+' denotes modulo 2 addition, allows the sinks to receive two packets where the first packet is either A or B and the second packet is $A + B$. To recover both A and B, the sink subtracts the first packet from the second one. For example, the sink on the leftside in Figure 2-3 receives A directly from the source on the leftside and $A + B$ from the middle link. Using these two packets, the sink can recover B by subtracting A from $A + B$; thus, correctly decode both packets of information.

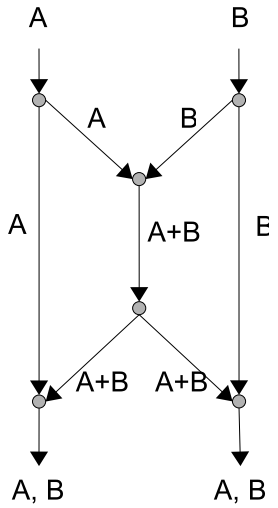

Figure 2-3: Using network coding on the butterfly network from [2]

2.1.1 Linear Network Coding

Now that we know network coding gives strictly greater throughput, we need to answer questions regarding what kind of coding to use; when we should use network coding and how much better it is compared to routing. There has been a lot of work recently to understand the advantages of network coding, and how to generate network codes that satisfy the rate demands of all the terminals in the network.

It all started with the work by Ahlswede et al. [2]. Reference [2] shows that coding within a network allows a source to mutlicast information at a rate approaching the smallest minimum cut between the source and any receiver, as the coding symbol

size approaches infinity. Li, Yeung and Cai [10] show that any solvable network with one source and multiple sinks (called *multicast network*) has a scalar linear solution over a sufficiently large finite field alphabet. In addition, [10] show that in multicast networks, linear coding suffices to achieve the optimum, which is the max-flow from the source to each sink. Subsequently, Koetter and Médard [8], noting the potential of linear network coding, present an algebraic framework for linear network coding to arbitrary networks and shows that a simple, linear code is sufficient to achieve capacity in the multicast problem.

As a result, there has been a huge emphasis on linear network coding due to its potential as a practical, simple code with near-optimum, if not optimum, performance. For instance, Ho et al. [7] propose a simple, practical capacity-achieving code. They propose that every node construct its linear code randomly and independently from all other nodes. This simple construction has been shown to achieve capacity with probability exponentially approaching 1 with the code length. This result indicate that linear network coding is a simple yet a very powerful tool. This has led Médard et al. [14] to conjecture that every solvble network has a linear solution over some finite field alphabet and vector dimensions. However, Dougherty et al. [5] provide a counterexample non-multicast network which is not solvable with linear coding. It is important to note that although [5] proves that linear network coding is not sufficient for all networks, linear network coding nevertheless is still a powerful tool. In particular, the example network in [5] is extremely complex and contrived, and for our general purpose, we can consider this kind of networks to be outliers, and thus, linear network coding is most of the time sufficient. Therefore, in this thesis, we shall focus on linear network coding and its benefit in a special set of networks called multicast switches (see Section 2.3) which will hopefully bring insight to that of general networks.

2.2 Graph Theory

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. A graph $G_1 = (V_1, E_1)$ is a subgraph of G if $V_1 \subseteq V$ and $E_1 \subseteq E$. A graph $G_2 = (V_2, E_2)$ is an *induced subgraph* of G if $V_2 \subseteq V$ and $(v_1, v_2) \in E_2$ if and only if $(v_1, v_2) \in E$. In addition, G_2 is often denoted as $G(V_2)$ and is said to be induced by V_2. The *complement* of graph G is a graph $\overline{G}$ on the same vertex set V such that two vertices of $\overline{G}$ are adjacent if and only if they are not adjacent in G. The *chromatic number* of a graph G is the smallest number of colors $\chi(G)$ needed to color the vertices of G so that no two adjacent vertices share the same color.

G is a *complete graph* if for every pair of vertices in V there exists an edge connecting the two, and V is called a *clique*. If for every pair of vertices in V there is no edge connecting the two, then V is said to be a *stable set*. G is a *hole* if it is a chordless cycle; G is called an *odd hole* if it is a hole of odd length at least 5. G is an *anti-hole* if its complement is a hole; G is an *odd anti-hole* if its complement is an odd hole. G is said to be *perfect* if for every induced subgraph of G, the size of the largest clique equals the chromatic number.

2.2.1 Stable Set Polytope

The *stable set polytope* $STAB(G)$ of a graph $G = (V, E)$ is the convex hull of the incidence vectors[1] x of these stable sets of the graph G. For a general graph G, it is NP-hard to compute the stable set polytope $STAB(G)$ and thus, a complete characterization of $STAB(G)$ in terms of linear inequalities is unknown.

However, several families of necessary conditions are known. One example is the *clique inequalities*:

$$\sum_{i \in Q} x_i \leq 1 \tag{2.1}$$

for all cliques Q in G. Clique inequalities of a graph say that the total weight on the vertices of maximal cliques must not exceed 1. Note that a incidence vector of a stable

[1]The incidence vector of G is a vector x whose entries are labeled with the vertices of G. If $x_i = 1$, then vertex i is in the set; otherwise, i is not part of the set.

set must satisfy all the clique inequalities since a stable set can only have at most one vertex from each clique in a graph. Thus, this shows that the clique inequalities are necessary conditions for the stable set polytope. The polytope described by these clique inequalities along with non-negativity constraints

$$x_i \geq 0 \tag{2.2}$$

for all nodes i of G is called the *fractional stable set polytope* $QSTAB(G)$. The fractional stable set polytope is often used as a canonical relaxation of $STAB(G)$. Note that, for most graphs, $STAB(G) \subsetneq QSTAB(G)$, since the clique inequalities are necessary but not sufficient conditions for stable set polytope. The two polytopes coincide precisely when G is *perfect*.

2.2.2 Perfect Graph

A *perfect graph* is a graph in which the chromatic number of every induced subgraph equals the clique number of that induced subgraph. It is not hard to see that in any graph, the clique number is a lower bound on the chromatic number, since all vertices in a clique must be assigned a distinct color in any proper coloring. Perfect graphs are those for which this lower bound is tight for all its induced subgraphs.

One of the important features of perfect graph is that many NP-hard graph problems become easy to solve on perfect graphs. For example, the graph coloring problem, maximum clique problem, maximum stable set problems as well as the stable set polytope problems are all known to be solvable in polynomial time for perfect graphs. This property of perfect graphs lends us a complete characterization of $STAB(G)$ in terms of linear inequalities; $STAB(G) = QSTAB(G)$ if and only if G is perfect, thus $STAB(G)$ is defined by the clique inequalities and the non-negativity constraints if and only if G is perfect.

We now state three well-known theorems that characterize perfect graphs.

Theorem 2.2.1 (Weak Perfect Graph Theorem [11]) A graph G is perfect if and only if its complement is perfect.

Theorem 2.2.2 (Strong Perfect Graph Theorem [15]) A graph G is perfect if and only if it contains no odd hole and no odd anti-hole.

Lemma 2.2.3 (Replication Lemma [11]) Let $G = (V, E)$ be a perfect graph and $v \in V$. Create a new vertex v' and join it to v and to all the neighbors of v. Then, the resulting graph G' is perfect.

It is not hard to imagine that there can be different degree of "perfectness" in a graph. We can consider two graphs G and H where both are not perfect but $STAB(G) \approx QSTAB(G)$ while $STAB(H) \ll QSTAB(H)$. In such a case, we would consider G to be "more perfect" than H. This observation gives rise to the need of a metric which measures how perfect a graph is. The *imperfection ratio* [6] was introduced precisely for this purpose.

2.2.3 Imperfection Ratio

In [6], the imperfection ratio $\mathsf{imp}(G)$ of graph G is defined as

$$\mathsf{imp}(G) = \min\{t : QSTAB(G) \subseteq t\ STAB(G)\}. \tag{2.3}$$

In essence, the imperfection ratio measures how much bigger the fractional stable set polytope $QSTAB(G)$ is relative to the stable set polytope $STAB(G)$. Note that for a perfect graph G, $\mathsf{imp}(G) = 1$. Therefore, $\mathsf{imp}(G) \geq 1$ for any graph G.

A useful bound on the imperfection ratio is presented in [6], which we reproduce below.

Proposition 2.2.4 (Gerke and McDiarmid [6]) For a graph G, if each vertex in G can be covered p times by a family of q induced perfect subgraphs, then $\mathsf{imp}(G) \leq \frac{q}{p}$.

We shall later revisit this notion of imperfectness of a graph when we study the rate regions of mutlicast switches in Chapter 4 and relate this notion to speedup in switches.

2.3 Multicast Switches

Multicast switches can be thought of as simple information networks where there are only sources and sinks, no intermediate nodes. Each source is connected to all sinks. In the most basic model, a switch acts as a router. After giving some definitions, we present on few different strategies used to service multicasts in a switch.

In a $K \times N$ input-queued multicast switch, there are K sources or *inputs* and N sinks or *outputs*. Each input can send a packet simultaneously to any given subset of the outputs where this subset of the outputs is called the *fanout set*. The *admissibility condition* requires that each node only send or receive at most one packet at any given time. In other words, an input can send a single packet to multiple outputs at once, but an input cannot send two different packets simultaneously; an output cannot receive two different packets simultaneously. The admissibility condition gives rise to the need for queues at the inputs as multiple packets may arrive at an input simultaneously. Thus, a set of $2^N - 1$ queues is maintained per input so that an input has separate queue for every fanout set. A diagram of an $K \times N$ switch is given in Figure 2-4.

A good definition of *time* is important in understanding the concept of *rate* and *speedup* (see Section 2.3.3) which are defined later in the thesis. There are two potential definitions for time units:

1. time it takes for a packet to travel from an input to an output;

2. minimum time between arrivals of two packets at an input.

In this thesis, the second definition is used to define an unit of time.

A *rate* specifies the number of packets that needs to be transferred from an input to a fanout set in one unit of time. A rate of 1/2, for example, means that the input has to send one packet over two time units.

A *flow* is a stream of packets that have common source and destination fanout set. It is represented by a 3-tuple (r, i, J) consisting of a rate r, an input i and a subset J of outputs corresponding to the destination set of the multicast stream. For

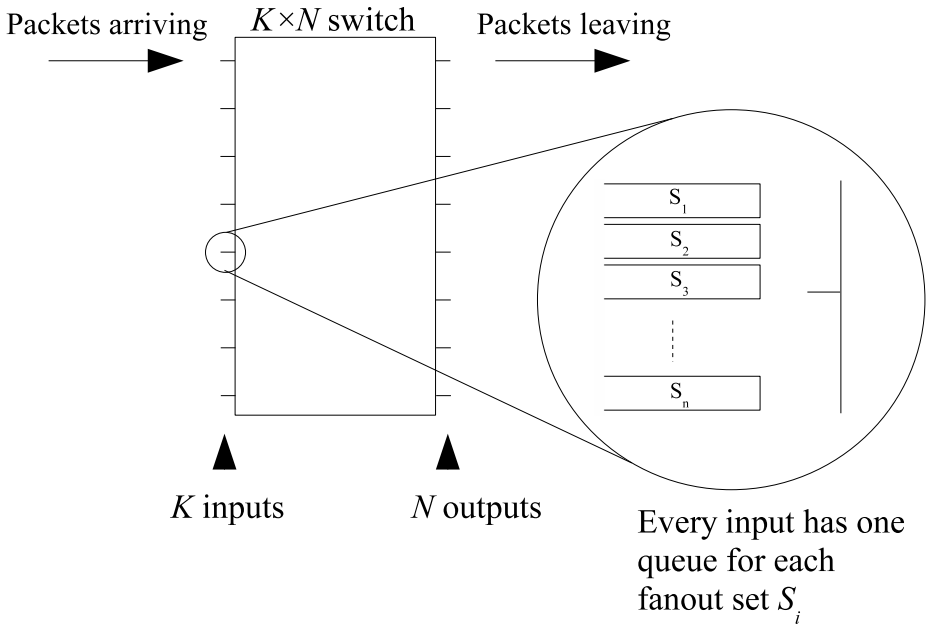

Figure 2-4: $K \times N$ input-queued multicast switch

example, in a 2×3 switch, we could have a flow $f = (1/2, 1, \{1, 2\})$ which is a stream of packets from input 1 to outputs 1 and 2 with rate of $1/2$. A *subflow* of flow (r, i, J) is a part of a flow from input i that goes to a particular output in J. Therefore, a subflow is a 4-tuple (r, i, J, j) consisting of a rate r, an input i, a subset of outputs J and one output $j \in J$. For instance, a flow $f = (1/2, 1, \{1, 2\})$ has two subflows associated with it: $f_1 = (1/2, 1, \{1, 2\}, 1)$ and $f_2 = (1/2, 1, \{1, 2\}, 2)$.

A *traffic pattern* is a collection of flows. A traffic pattern is called *admissible* if the total flow through each input or output does not exceed one – *i.e.* the inputs or outputs are not oversubscribed. A traffic pattern $\mathbf{r}$ is said to be *achievable* if there exists a switch schedule that can serve it. Intuitively, it may seem that an admissible traffic pattern is achievable; however, that is not the case. This intuition holds true for unicast traffic but not for multicast traffic.

Reference [13] shows that for unicast traffic, a switch can achieve 100% throughput – in the sense that switch can serve any admissible traffic pattern without causing the queues to grow unboundedly. Furthermore, Chang et al. [4] present a scheme, called the Birkhoff-von Neumann switch, that not only achieves 100% throughput

but also guarantees cell delay for unicast traffic in offline settings. The Birkhoff-von Neumann switch is based on a theorem that says any doubly stochastic matrix can be expressed as a convex combination of permutation matrices. Note that, any admissible unicast traffic pattern can be converted to a doubly stochastic matrix. Then, the doubly stochastic matrix is decomposed into permutation matrices, which in turn correspond to a switch state.

Sundararajan et al. [16] extend this Birkhoff-von Neumann approach to multicast switching. Using a graph theoretic formulation, they show that the rate region of multicast switching is precisely the stable set polytope of the traffic pattern's "conflict graph", which we shall discuss in Chapter 3. As a result, they show that the problem of deciding achievability in a multicast switch is equivalent to the membership problem for the stable set polytope of a graph, which is known to be NP-hard. In addition, Sundararajan et al. show that computing the offline schedule for multicast traffic, unlike unicast traffic, is hard, and is equivalent to fractional weighted graph coloring which is NP-hard in general.

Thus, many of the complexity and achievability results for unicast traffic do not extend to multicast traffic. One of the most important things to note is that the switch cannot achieve 100% throughput for multicast traffic. Even if a traffic pattern is admissible, depending on switch's capabilities, the switch may not be able to achieve the traffic pattern. For example, consider the traffic pattern shown in Figure 2-5. This traffic pattern consists of a broadcast flow $(2/3, 1, \{1, 2, 3\})$ which starts from input 1 and is destined to all the outputs at a of rate 2/3; a unicast flow $(1/3, 2, \{2\})$; and another unicast flow $(1/3, 2, \{3\})$. This traffic pattern shown in Figure 2-5 is admissible since every input and output has total rate of at most 1. However, a switch without any special capabilities requires at least two time units to serve two broadcasts and additional two time units to serve the remaining two unicasts. This implies that the switch is only servicing the broadcast at rate 2/4 and the unicasts at rate 1/4 each.

This observation that not all admissible traffic patterns are achievable raises the question of how much of the admissible rate region is actually achievable. To achieve

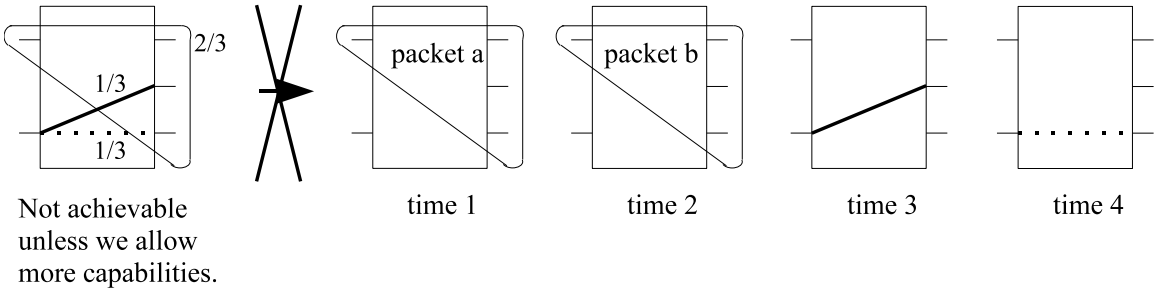

Figure 2-5: Admissible but not-achievable traffic pattern

those admissible but not achievable traffic patterns, what additional capabilities does a switch require? What capabilities of a switch is the most effective in increasing the achievable rate region to be at least the admissible rate region? In the next three sections, Section 2.3.1, 2.3.2 and 2.3.3, we present three capabilities – fanout splitting, intra-flow and inter-flow coding, and speedup – of a switch that increase its rate region.

2.3.1 Fanout Splitting

There are many ways in which a multicast switch can serve a multicast flow. The most simple method would be to serve all the multicast flow as if it was multiple unicast flows. For example, the packets of $f = (1/2, 1, \{1, 2\})$ could be "copied" into two separate unicasts $f_1 = (1/2, 1, \{1\})$ and $f_2 = (1/2, 1, \{2\})$. This scheme is inefficient since larger queues are required to service many multicasts. Further more, in some cases, this *copy strategy* converts an originally achievable traffic pattern into one that is inadmissible. For example, copying $f' = (1/2, 1, \{1, 2, 3\})$ into three unicasts will make three flows with rate 1/2 which overbooks input 1. The other extreme is to force the input to send the multicast packet to every output node in the fanout set simultaneously. This strategy is called *no-splitting*. However, this scheme can be restricting. As shown in Figure 2-5, if we use no-splitting strategy, the broadcast from input 1 at rate 2/3 requires two thirds of the time. This prevents input 2 from serving the two unicasts for two third of the time; otherwise, the outputs will be

oversubscribed. Then this leaves input 2 with one third of the time slots to serve two unicasts at rate 1/3 each, which is not possible.

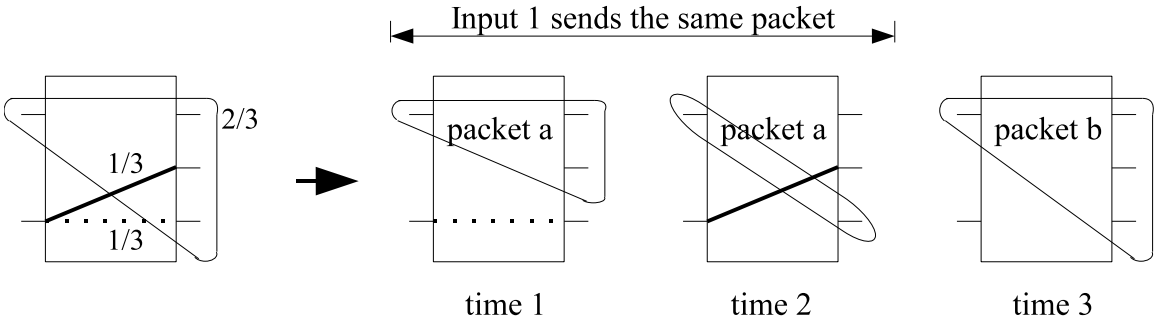

Figure 2-6: A traffic pattern which demonstrates the benefit of fanout-splitting

The middle ground between copying and no-splitting is *fanout-splitting*. Fanout-splitting allows the source to "store" packets and serve subsets of the fanout set at different points in time. Therefore, copying and no-splitting are two extreme cases of fanout-splitting: the first serves the fanout set by dividing it into subsets of size one, the latter serves it by not splitting at all. It is known that fanout-splitting achieves greater rate region than copying or no-splitting. For instance, Figure 2-5 shows a traffic pattern that cannot be satisfied by copying or no-splitting strategies, but with fanout-splitting this traffic pattern can be achieved as shown in Figure 2-6. Input 1 completes the broadcast of packet a over two time slots using fanout-splitting, while input 2 serves unicasts to the idle outputs over the two time slots. In the remaining third time slot, input 1 serves a second broadcast of packet b; thus, achieving the rate of 2/3 for the broadcast.

However, even with fanout-splitting, some traffic patterns are not achievable. Figure 2-7 gives an example of such case. This traffic pattern is very similar to that of in Figure 2-5, however, there is an additional unicast of rate 1/3 from input 2. In order for input 2 to complete all three unicasts, at every time slot, input 2 needs to be serving one of the unicasts. As a result, input 1 can not complete the broadcast of rate 2/3 even if it is allowed to use fanout-splitting.

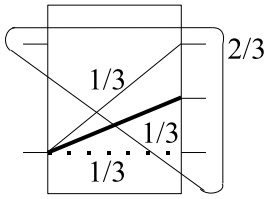

Figure 2-7: A traffic pattern that shows the limitation of fanout-splitting

2.3.2 Intra-flow and Inter-flow Coding

There are two types of coding: *intra-flow* coding and *inter-flow* coding. Intra-flow coding only encodes packets from the same flow while inter-flow coding can encode packets from the same flow as well as packets from different flows that originate from the same input. By definition, inter-flow coding has a greater rate region than that of intra-flow coding. In this thesis, we shall focus on the benefit of intra-flow coding; thus, in the remaining part of this thesis, unless specified otherwise, network coding means intra-flow coding.

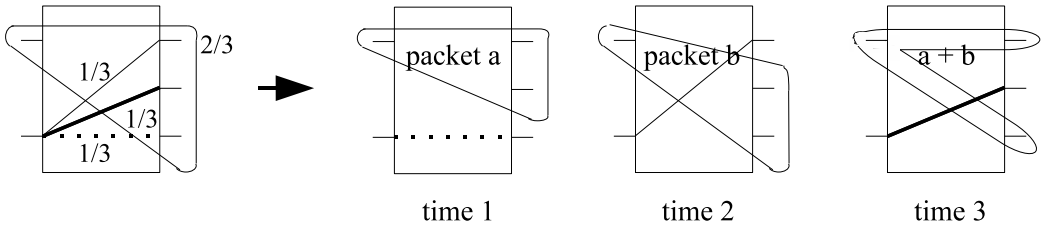

Figure 2-8: A traffic pattern that shows the benefit of coding

The benefit of intra-flow network coding can be seen in Figure 2-8, which illustrates a schedule that achieves the traffic pattern which we showed cannot be achieved using just fanout-splitting in Figure 2-7. Therefore, this shows that network coding rate region is greater than that of fanout-splitting. However, it is important to note that network coding does not achieve all admissible rate region. For instance, Figure 2-9 shows a traffic pattern which is admissible but not achievable even when network coding is allowed.

This observation brings into attention the question of how much of the admissible

rate region does network coding rate region actually achieve? We shall discuss in more detail these questions regarding the benefit of network coding in switches in Chapter 3.

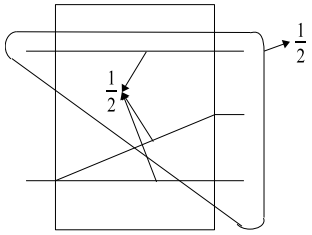

Figure 2-9: A traffic pattern which cannot be achieved by network coding

2.3.3 Speedup

Speedup allows a switch to operate on a different time scale than the outside world. A switch is said to have a *speedup* of s if the switching internally can transfer s packets over one time unit (as defined in Section 2.3). This means the switching fabric can go through s configurations within one time slot. In other words, during the time it takes for a packet to arrive at the switch, the switch can change its configuration s times. Speedup is usually achieved by parallelization of hardware.

It is important to note that with enough speedup, a switch can achieve any traffic pattern. For exmaple, in a $K \times N$ switch, if $s \geq K$ then any admissible traffic pattern r is achievable. Given any admissible traffic pattern, the switch can divide the traffic pattern so that each of the K inputs are separately served. Therefore, as shown in Figure 2-10, the switch will serve whatever traffic input 1 needs to send, then input 2, 3, and so forth. Since the switch has speedup of $s \geq K$, the switch can internally process the K inputs separately and still satisfy all the multicast requirements.

Therefore, the key question is what is the upper bound on speedup we need to achieve all admissible traffic pattern? From our example in Figure 2-10, we know that we can upper bound the speedup by K in a $K \times N$ swtich; however, can we find a better bound? In addition, as noted in Section 2.3.2, we know that network coding increases

throughput but not enough to cover the entire admissible rate region. However, we know that with enough speedup we know that any traffic pattern is achievable. Then, our next question is how much speedup does network coding replace? When does this happen? This question will be discussed in more detail in Chapter 4.

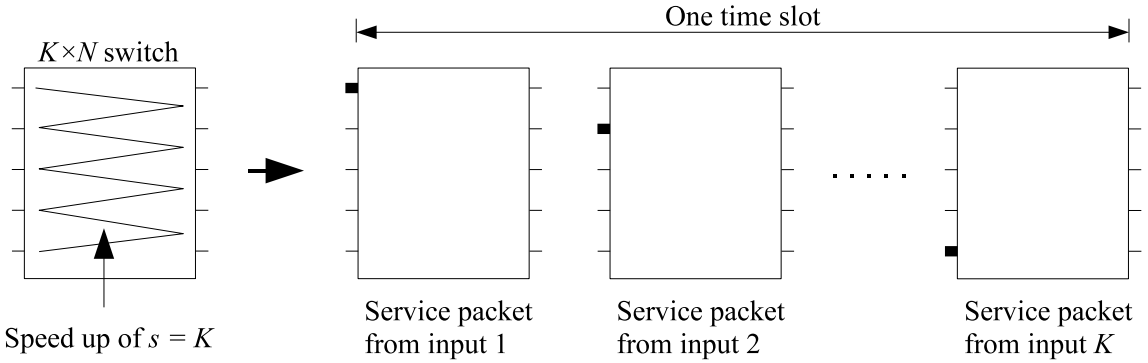

Figure 2-10: Speed up of $s = K$ for an $K \times N$ multicast switch

Chapter 3

Network Coding in Multicast Switches

In this Chapter, we demonstrate the benefit of network coding in multicast switches. In Section 3.1, we present a graph theoretic formulation of network coding in switches, which we shall use in Chapter 4 to obtain our main result. In Section 3.2, we present the increase in the rate region due to network coding in a multicast switch through examples and simulations. In Section 3.3, we study how network coding, even if it does not improve the rate, can decrease delay dramatically.

3.1 Conflict Graph

Let $\mathcal{N} = (V, E)$ be a directed acyclic graph which represents a network. The conflict graph $\mathcal{N}' = (V', E')$ is an undirected graph, corresponding to the network $\mathcal{N}$ is constructed as follows:

- For every link $l \in E$, create a set of nodes $n_{(l,s)}$ so that there is a one-to-one correspondence between all the possible states s of link l and nodes $n_{(l,s)}$.
- Connect two nodes $n_{(l,s)}$ and $n_{(l',s')}$ if assigning both state s to link l and state s' to link l' simultaneously is impossible. This implies that there is an edge between all pairs of $n_{(l,s)}$ and $n_{(l,t)}$ where $s \neq t$ since a link cannot be assigned

two different states simultaneously. Furthermore, given a set of inputs, a node can only output a function of those inputs. Thus, if the output link state is not compatible with the combination of input link states, we connect those states with an hyperedge.

Once we have constructed our conflict graph, a stable set represents a collection of states for links such that there is no conflict, i.e. it is possible to assign the set of states to the links in the network. Thus, a valid code in the network corresponds to a stable set, and any feasible rate can be achieved by time-sharing between the stable sets. This means that we can represent all achievable rate region by a convex hull of the stable sets - *i.e.* the stable set polytope of the conflict graph.

It is important to note that although this conflict graph formulation is elegant and easy to conceptualize, it has been noted in [18] that the size of a conflict graph grows exponentially with the number of possible states for each links. Furthermore, the problem of computing the stable set polytope of a graph is known to be NP-Hard as discussed in Section 2.2.1. Thus, we do not expect to find an algorithm that computes the stable set polytope or the rate region in polynomial time with respect to the size of the network. This implies that an algorithm that constructs a schedule by time-sharing between the stable sets cannot run in polynomial time, since the algorithm assumes the knowledge of the stable sets and thus, the description of the stable set polytope. As a result, running simulations using this algorithm on large network is impractical. This motivates us to look into the combinatorial and graph-theoretical tools to help us understand the benefit of network coding

3.1.1 Enhanced Conflict Graph

The *enhanced conflict graph* is a special kind of conflict graphs introduced by [18], which is used to characterize the rate region of multicast switches using network coding. A enhanced conflict graph $G = (V, E)$ is an undirected graph defined as follows:

- For every *subflow* create a nodes.

- A node representing subflow (r, i, J, j) is connected to all nodes whose corresponding subflows (r', i, J', j') belongs to other flows $(J \neq J')$ at the same input i. In addition, each subflow (r, i, J, j) is also connected to all subflows (r', i', J', j) that have the same output j.

The enhanced conflict graph is constructed such that the maximal cliques reflects the admissibility condition. For instance, the constraint that no input should send more than one unique packet at a time is represented by the edges connecting nodes corresponding to subflows (r, i, J, j) and (r', i, J', j') where $J \neq J'$. It is important to note that nodes representing subflows from the same flow, for example (r, i, J, j) and (r, i, J, j') where $j \neq j'$, are not adjacent. This is because two subflows from the same flow can be served simultaneously since input i can a single packet to multiple outputs. The second constraint that no output should receive more than one packet at a time is shown by the edges connecting nodes all subflows (r, i, J, j) and (r', i', J', j') where $j = j'$.

In addition to encoding the admissibility condition with cliques, the enhanced conflict graph also encodes information about achievable rate region. A stable set in an enhanced conflict graph represents a set of subflows that can be served simultaneously in a valid configuration of the switch. For instance, any subset of the subflows that belong to the same multicast flow form a stable set, and they can be served simultaneously.

Reference [17] shows that the stable set polytope and the fractional stable set polytope of an enhanced conflict graph are the rate region and the admissible rate region of a switch, respectively. In section 2.2, we showed that an explicit description of the fractional stable set polytope is known; thus, the admissible rate region of a switch is easy to compute. However, in section 2.2.1, we discussed the difficulty of computing the stable set polytope of a general graph, which in turn determines the difficulty of computing the rate region of a multicast switch. The key observation here is that if the enhanced conflict graph is perfect, then the problem of computing the achievable rate region of a switch becomes easy – the achievable rate region is the admissible rate region. In addition, as noted in section 2.2.3, the less "imperfect"

a conflict graph is, the closer the stable set polytope is to the fractional stable set polytope. Therefore, understanding and measuring the perfectness of the enhanced conflict graph is an elegant way of gaining insights into the benefit of network coding.

This graph-theoretic formulation helps us transform any given traffic pattern in a multicast switch into an enhanced conflict graph, and the properties of this graph can be used to derive insight on the rate regions of the switch as well as the benefit of network coding. A similar graph-theoretic formulation was used by Caramanis *et al.* [3] in the context of unicast traffic in Banyan networks.

Note that, for the case of fanout splitting without coding, [12] gave a characterization of the rate region as the convex hull of certain modified departure vectors. However, a graph-theoretic formulation of the same is not known. Therefore, network coding not only increases throughput, but also provides an insightful formulation that brings the problem to its combinatorial essence, which determines its difficulty.

As shown above, network coding provides us with a useful tool called the enhanced conflict graph, which gives us insight into the rate regions of the multicast switch. However, network coding is not only useful for our theoretical understanding. It also gives us gain in the practical sense – both in terms of throughput and delay. In the following two section, we shall discuss the benefit of network coding in terms of throughput (Section 3.2) and delay (Section 3.3).

3.2 Improvement in Rate

As noted in Section 2, we know that network coding increases the throughput of networks in general. In this section, we discuss how benefits and limitations of network coding in switches.

In Figure 3-1, we show a special traffic pattern in a $2 \times N$ switch, which demonstrates the benefit of intra-flow coding. At input 1, there is one broadcast flow with rate $1 - \frac{1}{N}$; at input 2, there is one unicast to every output with rate $\frac{1}{N}$.

This traffic pattern cannot be achieved with fanout-splitting alone. To show this, we note that at every time slot, one of the unicasts from input 2 has to be served, since

input 2 has total inflow of rate 1 and cannot be idle at any point in time. Therefore, input 1 needs at least two time slots to complete each of its broadcast. This means that input 2 requires a total of $2(1 - \frac{1}{N}) > 1$ time slots to finish if $N > 2$. Reference [17] shows that this traffic pattern is achievable with fanout-splitting alone if we allow a speedup of $1.5 - \frac{1}{N}$.

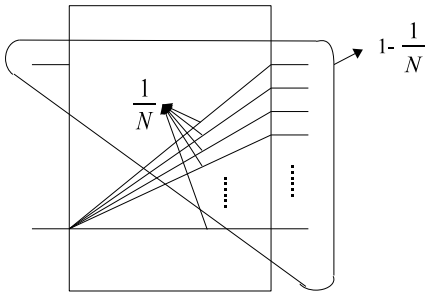

Figure 3-1: A traffic pattern which demonstrates the benefit of coding

However, this traffic pattern is achievable without speedup if network coding is allowed. The schedule looks similar to that shown in Figure 2-8. The code involves coding at input 1. Over N time slots, input 2 serves a unicast for one time slot to each output sequentially starting from output 1 to output N; thus, achieving the required rate of $\frac{1}{N}$ per unicast. While input 2 completes the unicasts, input 1 partially serves the broadcast requirement by sending a new packet at every time slot to all the outputs except the one occupied by input 2 during that time slot. During the last Nth time slot, input 1 combines all the previous $N - 1$ packets using an XOR operation and sends this "combined" packet to all available outputs. This schedule ensures that output N receives all $N - 1$ packets over N time slots; therefore, output N receives all the necessary packets to complete the broadcast of rate $1 - \frac{1}{N}$. In addition, the remaining outputs 1, 2, ..., $N - 1$ receive enough information to decode all $N - 1$ packets. Outputs 1, 2, ... , $N - 1$ receive $N - 2$ different packets and one "combined" packet. Each of these outputs, then, can decode the one remaining $N - 1$th packet by applying an XOR operation on all $N - 2$ packets and the "combined" packet. Thus, input 1 also successfully completes the broadcast requirement.

Polytope	Volume	Normalized Volume	Speedup to achieve P_{adm}
P_{adm}	4.921×10^{-9}	1	1
P_{intra}	4.686×10^{-9}	0.952	1.25
P_{fs}	4.613×10^{-9}	0.937	1.25
P_{nofs}	2.260×10^{-9}	0.460	1.67

Table 3.1: A comparision of the four schemes in a 2×3 switch

This example shows that network coding increases the rate region of a switch; thus, allowing the switch to serve higher rates. In the remaining of this section, we quantify the benefit of network coding by computing the rate regions. However, as noted in Section 3.1, computing the rate region (which is equivalent to computing the stable set polytope of a conflict graph) is NP-hard. As a result, we compute the rate regions of 2×3 switch only.

In a 2×3 switch, there are only three unicasts, three two-casts, and one broadcast from each of the two inputs. Therefore, the rate region is 14-dimensional polytope, which allows numerical computation to be feasible. We computed the stable set polytope of the enhanced conflict graph corresponding to a 2×3 switch to obtain the different rate regions as shown in Table 3.1.

The rate regions are compared in term of the volume of the polytope and the minimum speedup needed to achieve 100% throughput. The results are shown in Table 3.1. Here, P_{adm} refers to the admissible region; P_{intra} is the linear intra-flow network coding rate region; P_{fs} refers to the rate region with fanout-splitting only; and P_{nofs} is the rate region when fanout-splitting is not allowed.

The results show that coding does not outperform fanout-splitting by much. However, we should not interpret this result as such. Another way of looking at the two polytopes P_{intra} and P_{fs} is to just compare these two directly. From our previous example in Figure 3-1, we know that $P_{fs} \subsetneq P_{intra}$. So, we can ask what is the speedup needed for P_{fs} to achieve P_{intra}. For our 2×3 case, the speedup we need for the fanout-splitting region to achieve the network coding region is 1.1667. Therefore, this shows that network coding can give us a benefit equivalent to speedup of 1.1667.

The methodology we used to compute these values was to list all the stable sets

of the enhanced conflict graph using a greedy algorithm. Using these list of stable sets and a MATLAB packet called the multi-parametric toolbox [9], we computed the stable set polytope in terms of linear inequalities which in tern gave us the rate region. Once we have an explicit description of the rate regions, we used a software package known as Vinci [1] to compute the volume of the rate regions. The rate region of the case with fanout splitting but no coding was obtained using the characterization given by Marsan et al. [12]. The speedup required to achieve P_{adm} is equal to the minimum factor needed to expand the polytope such that it covers P_{adm}.

3.3 Improvement in Delay

In [17], Sundararajan et al. give an online scheduling algorithm for multicast switch with network coding. Although the rates of the various flows are not given beforehand, this algorithm uses the queue occupancy information to support the traffic without causing the queue to grow unboundedly. The algorithm is based on the idea that the stable set polytope of the conflict graph is the achievable rate region, and that any achievable rate can be decomposed into a convex combination of extreme points of the stable set polytope. Using this idea, Sundararajan et al. present the Maximum Weighted Stable Set (MWSS) algorithm, which computes the maximum weighted stable set on the enhanced conflict graph with queue lengths as weights. The MWSS algorithm then serves the traffic pattern represented by the maximum weighted stable set. Sundararajan et al. [17] show that the MWSS algorithm on the enhanced conflict graph achieves the entire rate region for multicast when intra-flow network coding is allowed.

However, the MWSS algorithm has no provision for packets to depart from the buffer, since the algorithm uses all packets the buffer has received so far in its code. Reference [17] presents an online algorithm called Finite Horizon MWSS algorithm which uses the MWSS algorithm with a batching scheme to stabilize the buffers. The basic idea is that packets that arrive at the switch is grouped into batches according to their arrival times. Then, we run the MWSS algorithm on a single batch as if

the packets in that batch are the only packets we shall receive. Once the batch has been served completely, the Finite Horizon MWSS algorithm takes a break to clear the backlog for that batch. After that, the batch is flushed out of the buffers, and the algorithm begins afresh with the next batch. For a more detailed analysis of the algorithm, see [17].

In this section, we study the effect of network coding in an online setting, through MATLAB simulations in a 4×3 switch. The scheduling algorithm we use is similar to the Finite Horizon MWSS algorithm, except, instead of computing the maximum weighted stable set which is known to be NP-hard, we use a simple randomized algorithm using the idea proposed in [19]. In each slot, we choose the best (or the one with maximum weight) of a constant number of randomly generated maximal stable sets, and the stable set that was used in the previous time slot. In this simulation, we grouped the packets into batches of size 1000.

We compared the performance of network coding with the case of fanout-splitting. For the fanout-splitting case, we use a similar randomized modification of the algorithm given in [12]. As previously mentioned, a graph theoretic formulation for the rate region of fanout-splitting is unknown; thus, instead of stable sets, we work with the modified departure vectors defined in reference [12].

The traffic pattern is chosen to be a combination of the example pattern used in Figure 2-7 weighted by a factor of $\frac{2}{3}\alpha$, and a pattern with uniform unicasts, each having a rate of 0.001α, where α represents the load factor. Thus, the traffic pattern consists of one broadcast from input 1, with rate of $\frac{4}{9}\alpha$. There are 3 unicasts, one to each output, from input 1, input 3, and input 4, each having a rate of 0.01α. From input 2, there is a unicast of rate $(\frac{2}{9} + 0.01)\alpha$. The arrivals are generated according to an *i.i.d* Bernoulli process independently for each flow.

The result of the simulation is shown in Figure 3-2. The figure shows the plot of delay vs. load for the randomized algorithm with and without coding. At light loads, network coding algorithm incurs a larger delay due to coding and decoding costs. When the traffic is light, fanout-splitting just relays the packets from input

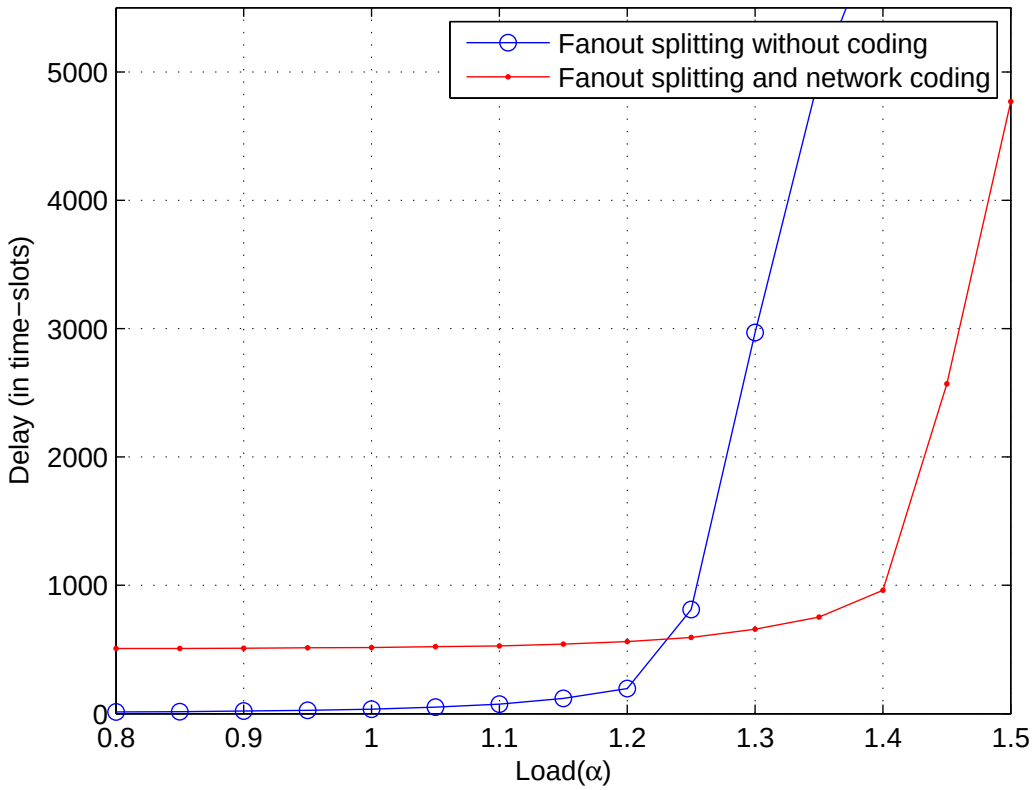

Figure 3-2: Delay vs. load plots with and without network coding in 4×3 switch

to output; however, network coding needs to wait until the entire batch is complete before the outputs can decode all the appropriate information it wants. As a result, we see that there is a consistent delay of 500 time slots for network coding at light loads. It is important to note that the delay of 500 is not an arbitrary delay. It is the average time slots each packet have to wait until it is decoded – and since our batch size is 1000, this average delay is 500.

The interesting part of our result is when the load is heavier. First we note that, for algorithm without network coding, the delay shoots up at a lower value of load α, as opposed to the network coding scheme. Therefore, in terms of throughput, the network coding scheme is clearly better. This empirically shows that network coding increases the rate region.

Furthermore, we note that in addition to gain in rate region, network coding gives

a significant benefit in terms of delay. We can consider the load beyond $\alpha = 1.4$ for instance. Here, the traffic pattern is clearly outside of the rate region for with and without network coding. Therefore, we would expect the delay for both schemes (with and without network coding) to shoot up, and it does. The part that interests us is that the difference in delay between the two schemes is significant. This shows that under heavy traffic, network coding is robust and, although the traffic pattern is beyond its rate region, it delivers the packets with much smaller delay than fanout-splitting scheme even when we take the coding and decoding cost into account.

Chapter 4

Network Coding for Speedup

In this chapter, we show the equivalence between network coding and speedup in multicast switches - *i.e.* network coding, which is usually implemented using software, can in many cases substitute speedup, which is often achieved by adding extra switch fabrics. Note that, we shall only consider intra-flow coding.

As discussed in Section 3.2, network coding improves throughput and can replace speedup. However, it is important to note that network coding cannot completely replace speedup. As noted above in Figure 3-1, there are situations where network coding replaces speedup; however, there are situations where speedup is still needed even if we allow network coding. For instance, recall the traffic pattern shown in Figure 2-9. At input 1, there is a broadcast flow and a unicast to output 1 with rate $\frac{1}{2}$ each; at input 2, there is one unicast flow to each output 2 and 3 with rate $\frac{1}{2}$. We showed that this traffic pattern cannot be achieved even when network coding is allowed; thus, we need speedup to achieve this traffic pattern. To explain this, we look into the enhanced conflict graph of this traffic pattern as shown in Figure 4-1.

In Figure 4-1, we show that the enhanced conflict graph for this traffic, where u_{ij} represents the unicast flow from input i to output j, and the vertex b_{ij} represents the broadcast subflow from input i to output j. The enhanced conflict graph contains an odd hole; therefore, it is not perfect. Thus, the achievable rate region is smaller than admissible rate region; the switch needs speedup even if we have network coding.

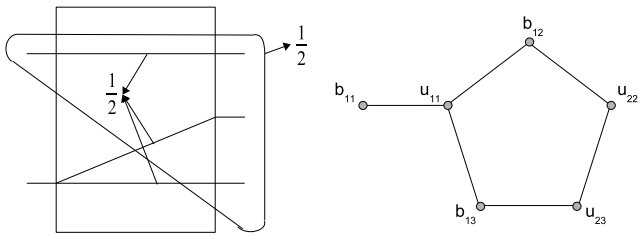

Figure 4-1: A traffic pattern which requires speedup and its enhanced conflict graph

The traffic pattern in Figure 4-1 requires speedup of 1.25 even with network coding. To understand how we figured out this value for the speedup needed to achieve this traffic pattern shown in Figure 4-1, we look into the description of the stable set polytope. There are many conditions, including the clique inequalities, that are known to be necessary conditions for the stable set poltyope. Another one of these condition is as follows:

$$\sum_{i \in H} x_i \leq \left\lfloor \frac{|H|}{2} \right\rfloor, \tag{4.1}$$

where H is an odd hole.

We observe that in Figure 4-1 each vertex in the odd hole represents a flow of rate 1/2. Therefore, the total weight on the odd hole is 5/2, which is the total rate the switch needs to serve to satisfy the sub-flows represented by the vertices in the odd hole. However, the right-hand side of inequality 4.1 is $\lfloor |H|/2 \rfloor = \lfloor 5/2 \rfloor = 2$. Hence, the smallest scaling factor such that the scaled vector lies inside the stable set polytope is 5/4. Therefore, a speedup of at least 5/4 is needed to serve this traffic pattern in a network coding switch.

On the other hand, it is not hard to see that this traffic pattern only requires speedup of at most 5/4 when network coding is allowed. To demonstrate, in Figure 4-2, we present a schedule for the traffic pattern shown in Figure 4-1. In this schedule, we note that the switch serves two packets for each flow, which requires a rate of 1/2 each. Therefore, if the traffic pattern was achievable with no speedup, the switch should serve this in 4 time slots; however, the switch actually uses 5 time slots. Therefore, this schedule assumes a speedup of 5/4, and the switch requires at most speedup of 5/4 to achieve this traffic pattern. Hence, this shows that the speedup

needed to achieve this traffic pattern is exactly 5/4.

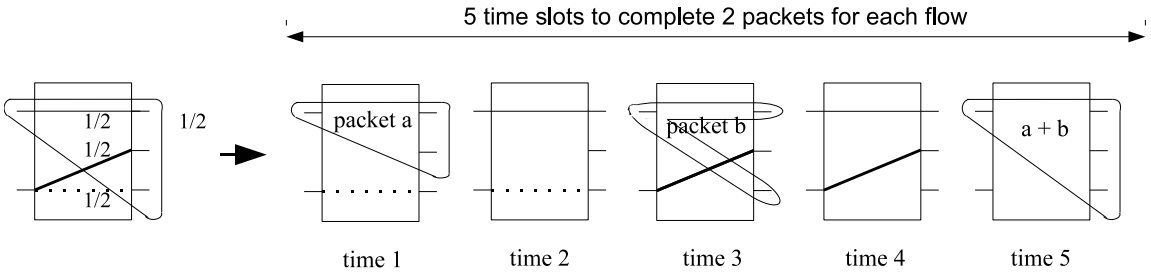

Figure 4-2: A traffic pattern that requires speedup in a network coding switch

Now, the important question we should answer is, how much speedup do we need to achieve this traffic pattern given a switch that uses network coding? Is there an upper bound on the speedup needed in a network coding switch to achieve 100% throughput? In this chapter, we study these questions and provide a upper bound on the minimum speedup $s_{\min}$ needed to achieve 100% throughput in a network coding switch.

Note that the traffic pattern in Figure 4-1 gives a lower bound on the speedup needed to achieve 100% throughput in a multicast switch using network coding. Therefore, $s_{\min} \geq 1.25$.

4.1 Rate Region of a Multicast Switch

In this section, we discuss how the stable set polytope of an enhanced conflict graph can translate to the rate region of a switch.

Let $\mathbf{r} \in \mathbb{R}^f$ be the *rate vector* of a traffic pattern that has f flows. Suppose that the total number of subflows in the pattern is m. Then, the *enhanced rate vector* $e(\mathbf{r}) \in \mathbb{R}^m$ corresponding to $\mathbf{r}$ is defined as:

$$e_{(i,J,j)}(\mathbf{r}) = \mathbf{r}_{(i,J)}, \text{ for all } j \in J.$$

Therefore, enhanced rate vector is just an extended version of the rate vector so that each flow is duplicated as many times as the number of its subflows. We use the

enhanced rate vector as *weights* for vertices of the enhanced conflict graph.

A traffic pattern $\mathbf{r}$ is said to be *achievable* if there exists a switch schedule that can serve it; it is called *admissible* if no input or output is oversubscribed. We also call the collection of all achievable and admissible vectors as the *achievable rate region* $\mathbf{R} \subseteq \mathbb{R}^f$ and *admissible rate region* $\mathbf{A} \subseteq \mathbb{R}^f$ respectively. For $\mathbf{r} \in \mathbf{R}$, we can construct a switch schedule, which can be viewed as a time sharing between valid switch configurations. In a conflict graph, a valid switch configuration corresponds to a stable set, and a switch schedule corresponds to a convex combination of stable sets of the conflict graph G. Therefore, if a rate vector $\mathbf{r} \in \mathbf{R}$, then $e(\mathbf{r}) \in STAB(G) \subseteq \mathbb{R}^m$. Thus, $\mathbf{R}$ is a projection of $STAB(G)$.

As mentioned in Section 2.2.1, a complete characterization of $STAB(G)$ in terms of linear inequalities is unknown for a general graph G. However, the fractional stable set polytope $QSTAB(G)$, a canonical relaxation of $STAB(G)$, can be described by the clique inequalities along with non-negativity constraints. In an enhanced conflict graph, the clique inequalities imply that no input nor any output may be overloaded. Therefore, in terms of the switch, [17] shows that the clique inequalities of the enhanced conflict graph correspond to the *admissibility conditions*. Thus, the $QSTAB(G)$ is the admissible rate region of the multicast switch. Therefore, if a rate vector $\mathbf{r} \in \mathbf{A}$, then $e(\mathbf{r}) \in QSTAB(G) \subseteq \mathbb{R}^m$, *i.e.* $\mathbf{A}$ is a projection of $QSTAB(G)$.

Here, we recall from Section 2.2.3 that imperfection ratio measures the "perfectness" of a graph G by computing the minimum value t such that $QSTAG(G) \subseteq t\ STAB(G)$. To be more precise, in [6], the imperfection ratio $\mathsf{imp}(G)$ of graph G is defined as $\mathsf{imp}(G) = \min\{t : QSTAB(G) \subseteq t\ STAB(G)\}$. As we noted above, in terms of a switch, the admissible region $\mathbf{A}$ and the achievable region $\mathbf{R}$ are projections of $QSTAB(G)$ and $STAB(G)$ respectively. Therefore, given the imperfection ratio $\mathsf{imp}(G)$ of an enhanced conflict graph G, we have $\mathbf{A} \subseteq \mathsf{imp}(G)\mathbf{R}$.

Therefore, we understand now that achievable rate region $\mathbf{R}$ as well as the admissible rate region $\mathbf{A}$ can be represented in terms $STAB(G)$ and $QSTAB(G)$ of the conflict graph G respectively. Note that, for most graphs, $STAB(G) \subsetneq QSTAB(G)$, since the clique inequalities are necessary but not sufficient conditions for a stable set

polytope. Thus, the admissible region is often a strict superset of the achievable rate region, which implies that it is not possible to achieve 100% throughput even with coding - we need speedup. We shall use this connection between the conflict graph and rate regions to draw insights into what kind of benefit network coding gives us in terms of speedup in the following section.

4.2 Imperfection Ratio Bounds Speedup

This section develops our main result, which relates speedup with imperfection ratio. From Section 2.3.3, we know that a switch is said to have a *speedup* s if the switching fabric can transfer packets at a rate s times the incoming and outgoing line rate of the switch. If we define a time slot to be the reciprocal of the line rate, then this means the switching fabric can go through s configurations within one time slot. With this definition, it is easy to see that a rate vector $\mathbf{r}$ is achievable with speedup s if and only if it is admissible and $\frac{1}{s}\mathbf{r}$ is within the rate region.

Note that the admissible and achievable rates correspond to $\mathbf{A}$ and $\mathbf{R}$ respectively. Then, $s_{\min} = \min\{s \mid \mathbf{A} \subseteq s\ \mathbf{R}\}$ is the *minimum speedup* required for the switch to achieve all admissible rates, *i.e.* it is the minimum of all s such that $\frac{1}{s}\mathbf{r}$ is within the rate region for all admissible rate vectors $\mathbf{r}$.

Note that, from out discussion in Section 4.1, we have shown that $\mathbf{R}$ and $\mathbf{A}$ are projections of $STAB(G)$ and $QSTAB(G)$ respectively. Therefore, the definition of imperfection ratio in Section 2.2.3 is very similar to that of minimum speedup above. This leads to our Corollary 4.2.1 below.

Corollary 4.2.1 *Given a traffic pattern, let G be its enhanced conflict graph and $s_{\min}$ be the minimum speedup required to achieve all admissible rates. Then, $s_{\min} \leq \mathsf{imp}(G)$.*

Note that the converse of Corollary 4.2.1 is not true. This is because $\mathbf{A}$ and $\mathbf{R}$ are projections of $QSTAB(G)$ and $STAB(G)$ such that the subflows corresponding to the same multicast flow have the same weight. As a result, $QSTAB(G) \subseteq \mathsf{imp}(G)STAB(G)$ implies the $\mathbf{A} \subseteq \mathsf{imp}(G)\mathbf{R}$, but $\mathbf{A} \subseteq s_{\min}\mathbf{R}$ may not imply $QSTAB(G) \subseteq s_{\min}STAB(G)$.

4.3 Bounds on Speedup for $2 \times N$ switch with unicasts and broadcasts

In this section, we apply Corollary 4.2.1 to $2 \times N$ switches using coding with traffic patterns consisting of unicasts and broadcasts only. We show that the minimum speedup needed for 100% throughput in this case is bounded by $\frac{3}{2}$. In this section, coding implies intra-flow coding, since enhanced conflict graphs handle intra-flow, not inter-flow, coding. The rest of this section is organized as follows. First, we give a description of the enhanced conflict graph for a $2 \times N$ switch with unicasts and broadcasts only. In Section 4.3.2, we show the bound on speedup of $\frac{3}{2}$.

4.3.1 Enhanced conflict graph for $2 \times N$ switch

Consider traffic patterns which consist of only unicasts and broadcasts in a $2 \times N$ switch. Then, the *enhanced conflict graph* $G_{2,N} = (V, E)$ has the following structure:

- The vertex set $V = U_1 \cup B_1 \cup U_2 \cup B_2$ where $U_i = \{u_{ij} \mid j \in [1, N]\}$ and $B_i = \{b_{ij} | j \in [1, N]\}$. The vertex u_{ij} represents the unicast flow from input i to output j, and the vertex b_{ij} represents the broadcast subflow from input i to output j. Therefore, U_i is the collection of the N unicast flows from input i, and B_i is the collection of the N sub-flows of the broadcast from input i.

- The edge set $E = E_1^u \cup E_2^u \cup E_1^b \cup E_2^b \cup E^o$ where:

$$E_i^u = \{(u_{ij}, u_{ik}) \mid j \neq k, j, k \in [1, N]\}$$
$$E_i^b = \{(b_{ij}, u_{ik}) \mid j, k \in [1, N]\}$$
$$E^o = \cup_{i \in [1,N]} E_i^o,$$

  where $E_i^o = \{(u_{ji}, u_{ki}), (b_{ji}, b_{ki}), (b_{ji}, u_{ki}), (b_{ji}, u_{ji}) \mid j \neq k, j, k \in [1, 2]\}$. Each edge set represents a different type of conflict. E_i^u represents conflicts among unicasts at input i; E_i^b represents conflict between any broadcast subflow and any unicast at input i; and E_i^o represents conflicts among all flows and subflows

at output i.

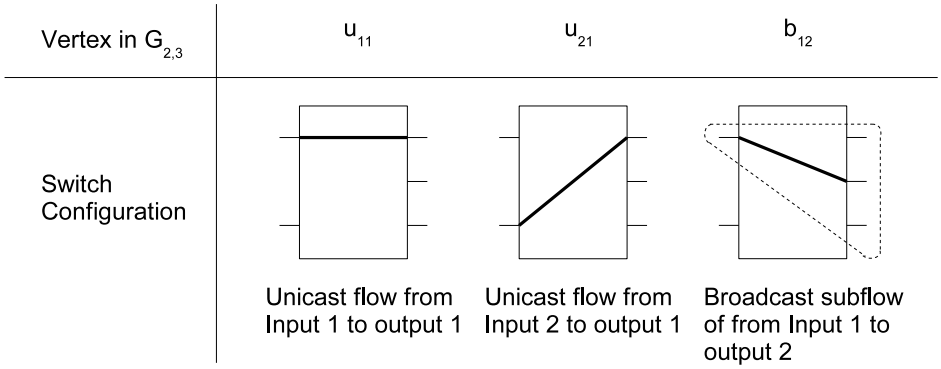

Figure 4-3: Switch configuration corresponding to u_{11}, u_{21}, and b_{12} in $G_{2,3}$

It is important to note that each vertex in $G_{2,N}$ represents a single flow or a subflow in a $2 \times N$. For example, u_{11} and u_{21} corresponds to a unicast traffic to output 1 from input 1 and input 2 respectively. The vertex b_{12} represents a partial service to output 2 of a broadcast from input 1. In Figure 4-3, we show the switch configuration corresponding to u_{11}, u_{21}, and b_{12} in a 2×3 switch.

The intuition behind conflict graph is that vertices, which represent flows that cannot be served simultaneously, are adjacent. For example, in Figure 4-4, we show an enhanced conflict graph for a 2×3 switch with unicasts and broadcasts only. There is an edge between u_{11} and b_{12} since they both represent flows serving input 1. There also exists an edge between u_{11} and u_{21} since they both serve output 1; however u_{21} and b_{12} are not adjacent since they do not conflict on the input nor the output side. Using this intuition, we can deduce that vertices u_{1j} for all j, representing unicast flows from input 1, are adjacent to each other due to input side conflict. This statement holds for u_{2j} for all j as well. Furthermore, we can deduce that b_{1j} for all j representing broadcast sub-flows from input 1 to be not adjacent to each other since broadcast sub-flows from the same flow can be served simultaneously. Therefore, we can think of $G_{2,N}$ consisting of two induced complete subgraphs $G_{2,N}(U_1)$ and $G_{2,N}(U_2)$ of size N and two induced stable sets $G_{2,N}(B_1)$ and $G_{2,N}(B_2)$ of size N.

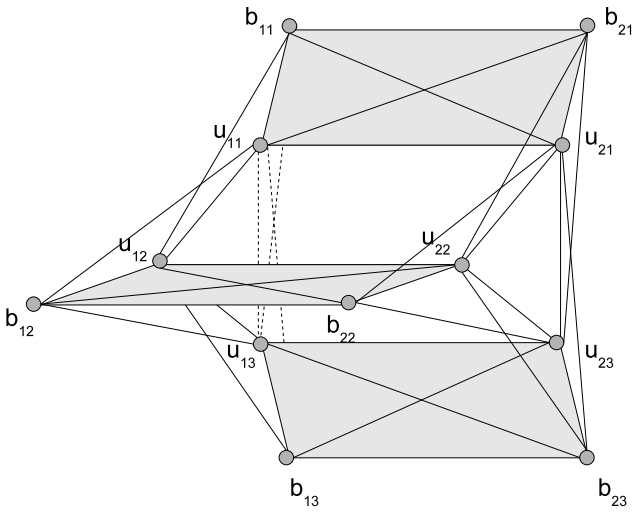

Figure 4-4: $G_{2,3}$ for a 2×3 switch with unicasts and broadcasts only

4.3.2 Speedup of $\frac{3}{2}$

To prove the upper bound of $\frac{3}{2}$ on speedup for a $2 \times N$ switch with restricted traffic of unicasts and broadcasts only, we use Proposition 2.2.4 introduced by Gerke and McDiarmid [6]. In the remaining of this section, we exhibit two types of induced subgraphs of $G_{2,N}$ that are perfect. Then, with Proposition 4.3.3, we use these induced perfect subgraphs to prove the upper bound on speedup.

Lemma 4.3.1 *Let $G_u = G_{2,N}(U_1 \cup U_2)$ be an induced subgraph of $G_{2,N}$. Then, G_u is perfect.*

Proof: G_u consists of two complete graphs $G_{2,N}(U_1)$ and $G_{2,N}(U_2)$. Therefore, the complement of G_u is a bipartite graph, which is known to be perfect. Then, by the Weak Perfect Graph Theorem, G_u is perfect as well. ■

Lemma 4.3.2 *Let $G_i = G_{2,N}(U_i \cup B_1 \cup B_2)$ where $i \in \{1, 2\}$ be an induced subgraph of $G_{2,N}$. Then, G_i is perfect.*

Proof: Assume that G_i is not perfect. Then, it must contain an odd-hole or an anti-hole as an induced subgraph. Suppose it has an odd hole, say H. Since $G_{2,N}(U_i)$ forms a complete graph, H cannot have more than two vertices from U_i; otherwise, the three vertices from U_i will form a triangle.

- *Case 1:* Consider the case when H has two vertices from U_i. Since all vertices of U_i are adjacent to every vertex in B_i, H cannot contain any vertex from B_i (otherwise, H contains a triangle graph). Then, it must be the case that H contains at least three vertices from B_j, $j \neq i$. However, vertices in B_2 form a stable set, which implies that H is not a hole.

- *Case 2:* Consider the case when H has only one vertex from U_i. Then, H can have at most two vertices from B_i. If H had more than two vertices from B_i, that would form a claw which cannot exist in a hole. Then, H must contain at least two vertices from B_j, $j \neq i$. However, no two vertices in B_j are adjacent, and thus, we cannot form a hole.

- *Case 3:* Consider the case when H does not contain any vertex from U_i. Then, H cannot be a cycle since there is no vertex in $G_{2,N}(B_1 \cup B_2)$ with degree greater than one.

Therefore, G_i does not contain an odd-hole. Then, G_i must contain an odd anti-hole, say A. For a vertex $v \in V$ to be in A, it must have degree greater than $|A| - 2 \geq 3$. Consider $b_{jk} \in B_j$ where $j \neq i$, $k \in [1, N]$. b_{jk} represents a broadcast subflow from input j to output k. Since no two nodes in B_j are adjacent to each other (*i.e.* there is no input conflict at input j), b_{jk} is adjacent to b_{ik} and u_{ik} only. Thus, no vertex $b_{jk} \in B_j$ can be in A. Then, A must be a subset of $U_1 \cup B_1$. However, $G_{2,N}(U_1 \cup B_1)$ is a *split graph*, a graph that can be partitioned into an stable set and a clique, which is known to be perfect. Thus, A cannot be a subset of $U_1 \cup B_1$.

This proves that G_i is perfect. ■

Proposition 4.3.3 $\mathsf{imp}(G_{2,N}) \leq \frac{3}{2}$.

Proof: Consider the three induced subgraph $G_u = G_{2,N}(U_1 \cup U_2)$, $G_1 = G_{2,N}(U_1 \cup B_1 \cup B_2)$ and $G_2 = G_{2,N}(U_2 \cup B_2 \cup B_1)$. An example of G_{uu} and G_{uub_2} for a 2×3 switch is shown in Figure 4-5. These three subgraphs are perfect by Lemma 4.3.1 and Lemma 4.3.2, and together they cover each vertex in $G_{2,N}$ twice. By Proposition 2.2.4, the claim follows. ■

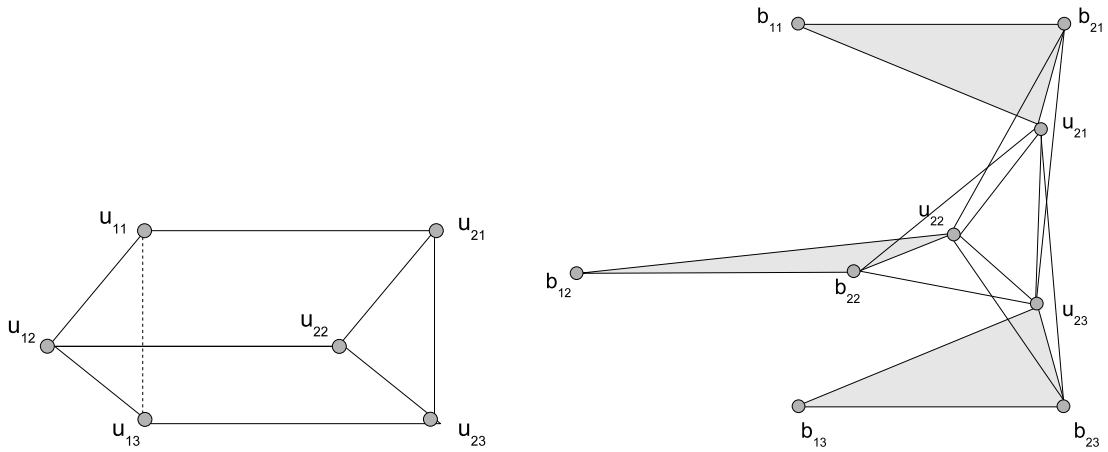

Figure 4-5: G_{uu} and G_{uub_2} for a 2×3 switch with unicasts and broadcasts only

4.4 Bounds on Speedup for $K \times N$ switch with unicasts and broadcasts

In this section, we generalize the result from Section 4.3. We apply Corollary 4.2.1 to $K \times N$ switches using intra-flow coding with traffic patterns consisting of unicasts and broadcasts only. We show that the minimum speedup needed for 100% throughput in this case is bounded by $\min(\frac{2K-1}{K}, \frac{2N}{N+1})$. In this section, coding implies intra-flow coding, since enhanced conflict graphs handle intra-flow, not inter-flow, coding. The rest of this section is organized as follows. First, we give a description of the enhanced conflict graph for a $K \times N$ switch. In Section 4.4.2 and 4.4.3, we show the two bounds on speedup of $\frac{2K-1}{K}$ and $\frac{2N}{N+1}$ respectively.

4.4.1 Enhanced conflict graph for $K \times N$ switch

Consider traffic patterns which consist only of unicasts and a broadcast per each input on a $K \times N$ switch. The basic idea behind conflict graph is that vertices representing flows that cannot be served simultaneously are adjacent. In such a case, the enhanced conflict graph $G_{K,N} = (V, E)$ has the following structure.

The vertex set $V = \left(\cup_{i\in[1,K]} U_i\right) \cup \left(\cup_{i\in[1,K]} B_i\right) = \left(\cup_{j\in[1,N]} U_j^o\right) \cup \left(\cup_{j\in[1,N]} B_j^o\right)$ where

$U_i = \{u_{ij} \mid j \in [1,N]\}$[1], $B_i = \{b_{ij} \mid j \in [1,N]\}$, $U_j^o = \{u_{ij} \mid i \in [1,K]\}$, and $B_j^o = \{b_{ij} | i \in [1,K]\}$. The vertex u_{ij} represents the unicast flow from input i to output j, and the vertex b_{ij} represents the broadcast subflow from input i to output j. Therefore, U_i and U_j^o are collections of the unicast flows from input i and to output j respectively. B_i and B_j^o are collections of the subflows of the broadcast from input i and to output j respectively.

The edge set $E = \left(\cup_{i\in[1,K]} E_i^u\right) \cup \left(\cup_{i\in[1,K]} E_i^b\right) \cup E^o$ where $E_i^u = \{(u_{ij}, u_{ik}) \mid j \neq k, j, k \in [1,N]\}$, $E_i^b = \{(b_{ij}, u_{ik}) \mid j, k \in [1,N]\}$, and $E^o = \cup_{i\in[1,N]} E_i^o$ where $E_i^o = \{(u_{ji}, u_{ki}), (b_{ji}, b_{ki}), (b_{ji}, u_{ki}) \mid j \neq k, j, k \in [1,K]\}$. Each edge set represents a different type of conflict. E_i^u represents conflicts among unicasts at input i; E_i^b represents conflict between any broadcast subflow and any unicast at input i; and E_i^o represents conflicts among all flows and subflows at output i.

As in the $2 \times N$ case, it is important to note that each vertex in $G_{K,N}$ represents a subflow in a $K \times N$ switch. The intuition behind a conflict graph is that vertices which represent flows that cannot be served simultaneously are adjacent. As shown in [17], if fanout splitting and network coding are allowed, the switch can simultaneously serve two or more subflows of the same broadcast flow and hence such subflows are not adjacent to each other. Therefore, from the input perspective, $G_{K,N}$ consists of K induced complete subgraphs $G_{K,N}(U_i)$ for unicasts from each input i, and K induced stable sets $G_{K,N}(B_i)$ for broadcasts from each input i; from the output perspective, $G_{K,N}$ consists of $2N$ induced complete subgraphs $G_{K,N}(U_j^o)$ and $G_{K,N}(B_j^o)$ for unicasts and broadcast subflows to each output j respectively.

Here, we note that conflict graph of a $K \times N$ multicast switch with unicasts and broadcasts traffic can be relaxed to that of unicasts and single multicast per input. This relaxation just removes vertices that represent broadcast subflows, which are not part of the multicast flow, from the conflict graph. Removing vertices from a graph cannot hurt the "perfectness" of the conflict graph. Therefore, any upper bound on the imperfection ratio of the conflict graph for unicasts and broadcasts bounds that of unicasts and single multicast per input.

[1] $j \in [1,N]$ means j can be integer from 1 to N.

4.4.2 Speedup of $\frac{2K-1}{K}$

In this Section, we give an upper bound on speedup for $K \times N$ switches. We present $2K-1$ induced perfect subgraphs of $G_{K,N}$ that cover the vertices V K times. Then, with Proposition 2.2.4, we then have $\frac{2K-1}{K}$ as an upper bound for speedup.

Lemma 4.4.1 Let $G_u = G_{K,N}(\cup_{i\in[1,K]}U_i)$ be an induced subgraph of $G_{K,N}$. Then G_u is perfect.

Proof: G_u is an enhanced conflict graph for unicast traffic. One may check that G_u is a line graph of a bipartite graph, which is known to be perfect [15]. ■

Lemma 4.4.1 also follows from the result in [13] which shows that 100% throughput can be achieved in a input-queued crossbar switch in the context of unicast traffic.

Lemma 4.4.2 Let $G_i = G_{K,N}\left((\cup_{j\in[1,K]}B_j)\cup U_i\right)$ for some $i \in [1, K]$ be an induced subgraph of $G_{K,N}$. Then G_i is perfect.

Proof: Assume that G_i is not perfect. So it must have an odd hole or odd anti-hole as an induced subgraph. Suppose it has an odd hole, say H. In G_i, any broadcast subflow, except the ones from input i, has no conflict on the input side. Suppose such a subflow were part of H, then both its neighbors in H will be due to output side conflicts. But in that case, the two neighbors will themselves conflict at the output, thereby forming a triangle. Since an odd hole cannot contain a triangle, we conclude that H cannot include any b_{jk} with $j \neq i$.

This means H must be an induced subgraph of $G_{K,N}(B_i \cup U_i)$. However, B_i induces a stable set, while U_i induces a clique. Therefore, $G_{K,N}(B_i \cup U_i)$ is a split graph[2] which is known to be perfect [15]. This contradiction shows that G_i cannot contain an odd hole H.

Suppose G_i contains an odd anti-hole, say A. This will happen if and only if $\overline{G_i}$ contains an odd hole H_A. Note that in $\overline{G_i}$, two vertices are connected if the corresponding subflows do not conflict. Now, H_A has to contain at least one unicast, say u_{ij}, since the broadcasts by themselves induce a perfect subgraph in $\overline{G_i}$ (they

[2] A *split graph* is a graph whose vertex set can be partitioned into a stable set and a clique.

induce the complement of a disjoint union of complete graphs, which is known to be perfect [15]). Now, u_{ij} in $\overline{G_i}$ is adjacent to any $b_{i'j'}$, where $i \neq i'$ and $j \neq j'$. Let b_{pq} and $b_{p'q'}$ be vertices adjacent to u_{ij} in H_A. Then, using the definition of $\overline{G_i}$, we can infer that $i \neq p \neq p' \neq i$ and $q = q' \neq j$. But this means, any vertex that is adjacent to b_{pq} is also adjacent to $b_{p'q'}$. Hence, H_A cannot be an odd hole.

This proves that G_i is perfect. ■

Using Lemmas 4.4.1 and 4.4.2, we derive our first upper bound on speedup in $K \times N$ multicast switches with traffic patterns consisting of unicasts and broadcasts only.

Proposition 4.4.3 $\mathsf{imp}(G_{K,N}) \leq \frac{2K-1}{K}$.

Proof: Consider the following collection of induced subgraphs: $K-1$ copies of G_u from Lemma 4.4.1 and G_i from Lemma 4.4.2 for all $i \in [1, K]$. We know that these subgraphs are all perfect. In addition, these subgraphs cover each vertex in $v \in G_{K,N}$ K times. For $v \in U_i$, each G_u and G_i covers v once. For $v \in B_i$, each G_i covers v. By Proposition 2.2.4, the claim follows. ■

4.4.3 Speedup of $\frac{2N}{N+1}$

The proof idea in this section is similar to that of Section 4.4.2. We present $2N$ induced perfect subgraphs of $G_{K,N}$ that cover the vertices V $N+1$ times, and then appeal to Proposition 2.2.4. However, unlike Section 4.4.2, here we change our focus from the input to output.

Lemma 4.4.4 Let $G^o_{1,i} = G_{K,N}(V_i)$ where $V_i = U^o_i \cup \left(\cup_{j\in[1,N]} B^o_j\right)$ be an induced subgraph of $G_{K,N}$. Then $G^o_{1,i}$ is perfect.

Proof: Assume that $G^o_{1,i}$ is not perfect. So it must have an odd hole or odd anti-hole as an induced subgraph. Suppose it has an odd hole, say H. Since $U^o_i \cup B^o_i$ forms a complete graph (known to be perfect), H must contain vertices of B^o_j, $j \neq i$. Suppose $b_{kj} \in B^o_j$ is part of H, then H contains at least two vertices of B^o_j. This is because, in $G^o_{1,i}$, b_{kj} has only one conflict on the input side; thus, neighbors of b_{kj} are

u_{ki} (input conflict) and B_j^o (output conflict). However, note that B_j^o itself forms a complete graph, therefore H contains at most two vertices of B_j^o. Thus, b_{kj} and $b_{k'j}$, $k \neq k'$ are in H. Then, u_{ki} and $u_{k'i}$ are in H. However, these four vertices form a cycle, thus $G_{1,i}^o$ cannot contain an odd hole H.

By the same argument as in the proof for Lemma 4.4.2, we can show that $G_{1,i}^o$ cannot contain an odd anti-hole. This proves our claim. ■

Lemma 4.4.5 Let $G_{2,i}^o = G_{K,N}(V_i)$ where $V_i = B_i^o \cup \left(\cup_{j\in[1,N]} U_j^o\right)$ be an induced subgraph of $G_{K,N}$. Then, $G_{2,i}^o$ is perfect.

Proof: $G_{2,i}^o$ is an enhanced conflict graph for unicast traffic in addition to all broadcast subflows to output i. Consider $b_{1i} \in B_i^o$ and $u_{1i} \in \cup_{i\in[1,K]} U_i$. In a $K \times N$ switch, b_{1i} and u_{1i} represent subflows from input 1 to output i, and thus conflict with the same set of subflows, *i.e.* neighbors of u_{1i} are neighbors of b_{1i}. In addition, b_{1i} and u_{1i} are in conflict. Therefore, by Replication Lemma (Lemma 2.2.3), we know that $G_{2,i}^o$ is perfect if $G_{K,N}(V_i \setminus \{b_{1i}\})$ is perfect. We can apply this argument repeatedly for each $b_{ji} \in B_i^o$, and deduce that if $G_{K,N}(\cup_{j\in[1,N]} U_j^o)$ perfect then $G_{2,i}^o$ is perfect. Note that from Lemma 4.4.1, we know that the enhanced conflict graph $G_u = G_{K,N}(\cup_{i\in[1,K]} U_i) = G_{K,N}(\cup_{j\in[1,N]} U_j^o)$ for unicast traffic is perfect. Therefore, $G_{2,i}^o$ is perfect. ■

Now, using Lemmas 4.4.4 and 4.4.5, we can derive an upper bound for speedup in $K \times N$ multicast switches with traffic patterns consisting of unicasts and broadcasts only.

Proposition 4.4.6 $\mathsf{imp}(G_{K,N}) \leq \frac{2N}{N+1}$.

Proof: Consider the following collection of induced subgraphs: $G_{1,i}^o$ and $G_{2,i}^o$ for all $i \in [1, N]$. By Lemmas 4.4.4 and 4.4.5, we know that these subgraphs are all perfect. In addition, these subgraphs cover each vertex in $v \in G_{K,N}$ $N+1$ times. By Proposition 2.2.4, the claim follows. ■

4.5 Summary

In this section, we introduce a simple graph theoretic bound on speedup needed to achieve 100% throughput in a multicast network coding switch using the concept of conflict graphs. We show that the imperfection ratio of the enhanced conflict graph gives an upper bound on speedup needed. We apply this result to $2 \times N$ switches with traffic patterns consisting of unicasts and broadcasts only to obtain an upper bound of 3/2. We generalize this approach to $K \times N$ switches and obtain an upper bound on speedup of $\min(\frac{2K-1}{K}, \frac{2N}{N+1})$.

Here, we would like to note that, for a $2 \times N$ switch, this gives a bound of 3/2 on speedup; however, we conjecture that the actual speedup required to achieve 100% throughput in a $2 \times N$ switch with traffic patterns consisting of unicasts and broadcasts only is exactly 5/4. We have already shown in the beginning of this chapter that the speedup needed to achieve 100% throughput is at least 5/4. Thus, if this conjecture is true, then it shows that in a $2 \times N$ switch, the "worst" traffic pattern, such as Figure 4-1, induces an enhanced conflict graph that contains an odd hole of size 5.

We have verified this conjecture using a computer for $N = 3, 4$ and 5. If this conjecture is true, then we have a tight bound on the minimum speedup needed to achieve 100% throughput in a $2 \times N$ switches with traffic patterns of unicasts and broadcasts only. This is because Figure 4-1 shows that we need speedup of at least 5/4 to achieve the shown traffic pattern with network coding. Therefore, if this conjecture if shown to be true, then speedup of 5/4 is the exact amount of speedup we need to achieve 100% throughput in this special case.

In summary, by allowing network coding in multicast switches, we get not only an insightful characterization of the speedup needed for 100% throughput, but also a gain in throughput, delay, and speedup. We have shown that network coding, which is usually implemented using software, can substitute speedup, which is often achieved by adding extra switch fabrics.

Chapter 5

Conclusions and Future Directions

This chapter summarizes the work presented in this thesis. In this thesis, we explored the questions regarding the benefit of network coding in multicast switch in terms of throughput, delay, and speedup. Although network coding includes coding schemes with any arbitrary functions, we focus our attention to linear network coding. This is because, for the most part, linear network coding is sufficient to achieve capacity in the multicast problem. This means that linear network coding gives us simplicity in code, in addition to the performance gain that we want from coding. In addition, using linear network coding allows us to utilize the graph theoretic formulation called the conflict graph from [18], which is an insightful formulation that brings the problem to its combinatorial essence, which determines its difficulty.

The main contribution of this thesis are:

- We show that network coding increases throughput and decreases delay. Network coding increases the rate region of a switch, and allows the switch to serve traffic patterns which it could not have otherwise. In addition, we show that network coding allows the switch to be robust even when the traffic demand is beyond its capacity – resulting in a much smaller delay compared to fanout-splitting under heavy load.

- We show that network coding can in many cases substitute speedup. The important point to keep in mind is that speedup is attained using parallelization

of hardware; while network coding is a software feature, which is easier to update and cheaper to implement.

- We provide a simple graph-theoretic upper bound on speedup to achieve 100% throughput in a network coding multicast switch using the concept of conflict graphs. We show that the imperfection ratio of the enhanced conflict graph gives an upper bound on speedup. We apply this result to $K \times N$ switches with traffic patterns consisting of unicasts and broadcasts only to obtain an upper bound of $\min(\frac{2K-1}{K}, \frac{2N}{N+1})$. For a $2 \times N$ switch, this gives a bound of 3/2 on speedup; however, we conjecture that the actual speedup required to achieve 100% throughput in a $2 \times N$ switch with traffic patterns consisting of unicasts and broadcasts only is 5/4. We have verified this conjecture using a computer for $N = 3, 4$ and 5.

In summary, by allowing network coding in multicast switches, we get not only an insightful characterization of the speedup needed for 100% throughput, but also a gain in throughput, delay, and speedup. We have shown that network coding, which is usually implemented using software, can substitute speedup, which is often achieved by adding extra switch fabrics. This thesis presents a graph theoretic approach to quantify the minimum speedup needed to achieve 100% throughput. This new formulation helps us better understand the problem and enables us to use combinatorial and graph theoretic results to measure the benefit of network coding in switches.

5.1 Future Directions

One of the most important remaining tasks is to expand this result from switches to that of general networks. However, as mentioned previously, the rate regions of general networks cannot be explicitly characterized as it has been shown to be NP-hard. Therefore, we need to use what we have learned from this thesis to obtain better descriptions or approximations of the rate regions of general networks. The end goal is to approximate difficult capacity region problems, create schedules from

such approximations, and eventually better understand the benefit of network coding in general networks.

There are several topics that we have considered for further research, as stepping stones to our ultimate goal. In this section, we discuss three possible lines of future work.

5.1.1 Inter-flow network coding

There are two types of coding: *intra-flow* coding and *inter-flow* coding. Intra-flow coding only encodes packets from the same flow while inter-flow coding can encode packets from the same flow as well as packets from different flows that originate from the same input. By definition, inter-flow coding has a greater rate region than that of intra-flow coding; however, in this thesis, we focused on the benefit of intra-flow coding. Therefore, a natural extension to this thesis is to study the benefit of inter-flow network coding.

When does inter-flow coding help? How can we measure it? Is there a graph theoretic formulation for inter-flow coding, as enhanced conflict graph is for intra-flow coding? The answers to these questions will allow us to truly assess the value of inter-flow coding.

5.1.2 Schedule

It is known that if a traffic pattern r is achievable then there exists a set of stable sets of the conflict graph whose convex combination is equal to r:

$$r = \sum_i \phi_i S_i, \tag{5.1}$$

where S_i is a stable set of the conflict graph and $\phi_i \in [0, 1]$ such that $\sum_i \phi_i = 1$. $< \phi_i >$ in Equation 5.1 is called a *schedule* and the value of ϕ_i gives the fraction of time the switch needs to be in state S_i.

There are two ways to measure the "goodness" of a schedule $< \phi_i >$ for a traffic

pattern $r = \sum_i \phi_i S_i$:

1. Schedule length: It is the time it takes to complete one round of the schedule. This is equal to the sum of ϕ_i. If $\sum \phi_i < 1$, then the switch is idle for $1 - \sum \phi_i$ fraction of the time; thus, shows that the switch can serve heavier traffic patterns. Therefore, we can consider a shorter schedule length as an indicator of higher efficiency in transferring packets from the inputs to the outputs.

2. Number of configurations: This is sometimes also called the *switch entropy*. This is the number of states S_i with $\phi_i > 0$. This is the number of different switch configurations needed to complete a round of the schedule. It is often the case that there is some cost associated with changing the configuration of a switch (which could conceivably be much larger than the time it takes to transfer a packet from an input to an output), then a schedule with fewest states could be the most efficient one.

Both of these definitions are important in understanding the different aspects of the strategies used. However, care is needed in choosing the algorithm to compute the schedule $< \phi_i >$ as the value of schedule length and number of configurations depends greatly on the schedule chosen. It is known that there can exist more than one convex combination of extreme points that equal an internal point in a polytope; thus, there can be different schedules for a single achievable traffic pattern.

For future work, we can ask how network coding affects the "goodness" of a schedule compared to fanout splitting. Does it reduce schedule length? If so, by how much? Does network coding reduce the number of configurations in a schedule? If that is the case, is the benefit in terms of schedule length and number of configuration large enough to justify the coding/decoding cost associated with network coding? These are few of the questions we can explore in this line of work.

5.1.3 Inheritance of speedup

Switches are often built up from smaller switches – *i.e.* switches are built recursively. For example, Banyan networks of size $2^m \times 2^m$ is built using two smaller Banyan

networks of size $2^{m-1} \times 2^{m-1}$ and 2^{m-1} switches of size 2×2, as shown in Figure 5-1.

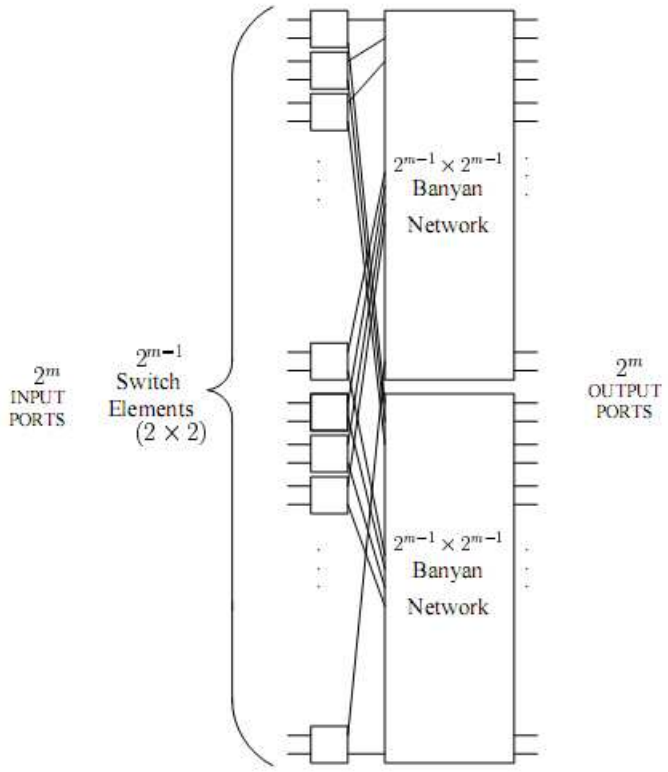

Figure 5-1: $2^m \times 2^m$ Banyan Network from [3]

Now assume that we need a speedup of s to achieve 100% throughput in a $2^{m-1} \times 2^{m-1}$ Banyan network with network coding. Then, how does the speedup needed in $2^m \times 2^m$ Banyan network relate to s? It is not hard to observe that we need at least speedup of $2s$ to achieve 100% throughput in $2^m \times 2^m$ Banyan network. This is because if input 1 and input 2 (which both use the first of the 2^{m-1} switch elements) want to send packets to output 1 and output 2 respectively, then the link connecting the first switch element to the upper $2^{m-1} \times 2^{m-1}$ Banyan network has to transfer both packets in a single time slot. This requires this link to have at least speedup of 2; thus, doubling the incoming traffic to the first input of the upper $2^{m-1} \times 2^{m-1}$ Banyan network. However, note that the $2^{m-1} \times 2^{m-1}$ Banyan network needs a speedup of s to achieve 100% throughput; thus, when the incoming traffic is doubled, it needs at least twice the original speedup to sustain the traffic pattern. Hence, $2^m \times 2^m$ Banyan

network needs a speedup of $2s$.

As seen above, there seems to be some kind of "inheritance" of speedup as we build switches out of smaller switches. So, for different switch architecture, how is speedup inherited? Can we get a better bound on speedup for Banyan networks? Reference [3] studies this problem in Banyan networks of size up to 8×8 in the context of unicast traffic. It would be an interesting problem to understand how performance and speedup are affected as we rely on smaller components, which themselves need speedup to achieve 100% throughput, to construct bigger switches. This in turn will help us understand how the benefit of network coding propagates as we build a switch from smaller components.

Bibliography

[1] Vinci - computing volumes of convex polytopes. *http://www.lix.polytechnique.fr/Labo/Andreas.Enge/Vinci.html*.

[2] Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li, and Raymond W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46:1204–1216, 2000.

[3] Constantine Caramanis, Michael Rosenblum, Michel X. Goemans, and Vahid Tarokh. Scheduling algorithms for providing flexible, rate-based, quality of service guarantees for packet-switching in banyan networks. In *Proceedings of the Conference on Information Sciences and Systems*, pages 160–166, 2004.

[4] Cheng-Sheng Chang, Duan-Shin Lee, and Yi-Shean Jou. Load balanced birkhoff-von neumann switches. In *Proceedings of IEEE Workshop on High Performance Switching and Routing*, 2001.

[5] Randall Dougherty, Christopher Freiling, and Ken Zeger. Insufficiency of linear coding in network information flow. *IEEE Transaction on Information Theory*, 51:2745–2759, 2005.

[6] Stefanie Gerke and Colin McDiarmid. Graph imperfection i, ii. *Journal of Combinatorial Theory, Series B*, 2001.

[7] Tracey Ho, Muriel Médard, Ralf Koetter, Michelle Effros, Jun Shi, and David R. Karger. A random linear coding approach to mutlicast. *IEEE Transaction on Information Theory*, 52:4413–4430, 2006.

[8] Ralf Koetter and Muriel Médard. An algebraic approach to network coding. *IEEE/ACM Transaction on Networking*, 11:782–795, 2003.

[9] Michal Kvasnica, Pascal Grieder, and Mato Baotic. Multi-parametric toolbox. *http://control.ee.ethz.ch/mpt/*, 2004.

[10] Shuo-Yen Robert Li, Raymond W. Yeung, and Ning Cai. Linear network coding. *IEEE Transaction on Information Theory*, 49:371–381, 2003.

[11] László Lovász. Normal hypergraphs and the perfect graph conjecture. *Discrete Mathematics*, 2:253–267, 1972.

[12] Marco Ajmone Marsan, Andrea Bianco, Paolo Giaccone, Emilio Leonardi, and Fabio Neri. Multicast traffic in input-queued switches: Optimal scheduling and maximum throughput. 11:465–477, June 2003.

[13] Nick McKeown, Venkat Anantharam, and Jean Walrand. Achieving 100% throughput in an input-queued switch. In *Proceedings of IEEE INFOCOM*, 1996.

[14] Muriel Médard, Michelle Effros, Tracey Ho, and David R. Karger. On coding for non-mutlicast networks. In *Proceedings of the 41st Annual Allerton Conference on Communication Control and Computing*, Monticello, Illinois, October 2003.

[15] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.*

[16] Jay Kumar Sundararajan, Supratim Deb, and Muriel Médard. Extending the birkhoff-von neumann switching strategy for multicast – on the use of optical splitting in switches. *IEEE Journal on Selected Areas in Communications - Optical Communications and Networking Series*, 2007.

[17] Jay Kumar Sundararajan, Muriel Médard, MinJi Kim, Atilla Eryilmaz, Devavrat Shah, and Ralf Koetter. Network coding in a multicast switch. In *Proceedings of IEEE INFOCOM*, 2007.

[18] Jay Kumar Sundararajan, Muriel Médard, Ralf Koetter, and Elona Erez. A systematic approach to network coding problems using conflict graphs. In *Proceedings of the UCSD Workshop on Information Theory and its Applications*, San Diego, CA, February 2006.

[19] Leandros Tassiulas. Linear copmlexity algorithms for maximum throughput in radio networks and input queueds switches. In *Proceedings of IEEE INFOCOM*, volume 2, pages 533–539, 1998.